CSDL-T-1251

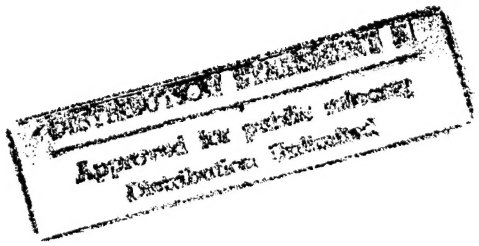
FAULT TOLERANT DESIGN USING SINGLE AND MULTICRITERIA GENETIC ALGORITHM OPTIMIZATION

by

Jason R. Schott

May 1995

Master of Science Thesis
Massachusetts Institute of Technology



DRAPER LABORATORY

The Charles Stark Draper Laboratory, Inc.
555 Technology Square, Cambridge, Massachusetts 02139-3563

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1995	

4. TITLE AND SUBTITLE
Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization

5. FUNDING NUMBERS

6. AUTHOR(S)
Jason R. Schott, 2Lt.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Students Attending:
Massachusetts Institute of Technology

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA
95-039

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTNEMT OF THE AIR FORCE
AFIT/CI
2950 P STREET, BDLG 125
WRIGHT-PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW AFR 190-1
Distribution Unlimited
BRIAN D. GAUTHIER, MSgt, USAF
Chief Administration

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

14. SUBJECT TERMS	15. NUMBER OF PAGES 200
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Fault Tolerant Design using Single and Multicriteria Genetic Algorithm Optimization

Jason R. Schott, Second Lieutenant
United States Air Force

1995

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for
the degree of Master of Science in Aeronautics and Astronautics
at the Massachusetts Institute of Technology

Abstract

This thesis incorporates a mixed discrete/continuous parameter genetic algorithm optimization capability into the Design Optimization/Markov Evaluation (DOME) program developed by the Charles Stark Draper Laboratory of Cambridge, Massachusetts. DOME combines the merits of Markov modeling and the Optimal Design Process to generate a systematic framework for system design with realistic reliability and cost analyses. The addition of genetic algorithms expands the design problem domain to include discrete parameter problems, which current optimization methods continue to struggle with.

A new variant of the genetic algorithm called the steady-state genetic algorithm is introduced to eliminate the idea of distinct generations. Functional constraints are dealt with by ingenious use of the function information contained in the genetic algorithm population. The optimal genetic algorithm parameter settings are investigated, and the genetic algorithm is compared to the Monte Carlo method and the Branch and Bound method to show its relative utility in optimization. This research shows that a single criterion genetic algorithm can be expected to outperform other methods in efficiency, accuracy, and speed on problems of moderate to high complexity.

The work then extends to multicriteria optimization, as applied to fault tolerant system design. A multicriteria genetic algorithm is created as a competitive means of generating the efficient (Pareto) set. Method parameters such as cloning, sharing, domination pressure, and population variability are investigated. The method is compared to the ε-constraint multicriteria method with a steady-state genetic algorithm performing the underlying single-criterion optimization. This research shows that a genetic algorithm using dominance as a selection criterion exhibits excellent performance for efficient set generation.

(200 pages)

Fault Tolerant Design using Single and Multicriteria Genetic Algorithm Optimization

by

Jason R. Schott, 2Lt, USAF

B.S., United States Air Force Academy (1993)

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1995

Signature of Author _____
Department of Aeronautics and Astronautics
May 1995

Approved by _____
Dr. Andrei L. Schor
Thesis Supervisor, C.S. Draper Laboratory

Certified by _____
Professor Wallace E. Vander Velde
Thesis Supervisor, Professor of Aeronautics and Astronautics

Accepted by _____
Professor Harold Y. Wachman
Chairman, Department Graduate Committee

Fault Tolerant Design using Single and Multicriteria Genetic Algorithm Optimization

by

Jason R. Schott, Second Lieutenant
United States Air Force

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for
the degree of Master of Science in Aeronautics and Astronautics
at the Massachusetts Institute of Technology

Abstract

This thesis incorporates a mixed discrete/continuous parameter genetic algorithm optimization capability into the Design Optimization/Markov Evaluation (DOME) program developed by the Charles Stark Draper Laboratory of Cambridge, Massachusetts. DOME combines the merits of Markov modeling and the Optimal Design Process to generate a systematic framework for fault tolerant system design with realistic reliability and cost analyses. The addition of genetic algorithms expands the permissible design problem domain to include discrete parameter problems, which current optimization methods continue to struggle with.

A new variant of the traditional genetic algorithm called the steady-state genetic algorithm is introduced to eliminate the idea of distinct generations. Functional constraints are dealt with by ingenious use of "fitness penalty" that capitalizes on the function information contained in the genetic algorithm population. The optimal genetic algorithm parameter settings are investigated, and compared to those of earlier work in this area. The genetic algorithm is compared to the Monte Carlo method and an adapted form of the Branch and Bound optimization method to show its relative utility in the optimization field. This research shows that a single criterion genetic algorithm can be expected to outperform other methods in efficiency, accuracy, and speed on problems of moderate to high complexity.

The work then extends to multicriteria optimization, as applied to fault tolerant system design. A multicriteria genetic algorithm is created as a competitive means of generating the efficient (Pareto) set. Method parameters such as cloning, sharing, domination pressure, and population variability are investigated. The method is compared to the ε-constraint multicriteria method with a steady-state genetic algorithm performing the underlying single-criterion optimization. This research shows that a genetic algorithm using dominance as a selection criterion exhibits excellent performance for efficient set generation.

Thesis Supervisor: Professor Wallace E. Vander Velde
Thesis Supervisor: Dr. Andrei L. Schor

Acknowledgments

I would like to express my gratitude to the people who made this effort both challenging and rewarding.

Andrei Schor, for providing the oversight to keep me on track and the freedom to follow the paths I thought best.

Professor Vander Velde, for asking the tough questions necessary to expand my thoughts and for his efforts in following my erratic progress.

Emily Schott, for being my wife and for supporting me even though I was never home.

Kent Engebretson, for allowing me to vent frustration and for punishing my body in the gym on a regular basis.

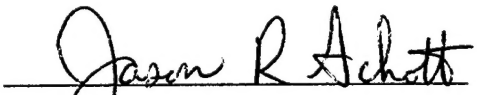
The Boys of 13 Bigelow, who made the first semester a most memorable one.

Jesus, my God and my Savior, who gives me the strength to push on.

This thesis was prepared at the Charles Stark Draper Laboratory, Incorporated under an internal development contract.

Publication of this thesis does not constitute approval by the Draper Laboratory of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

I hereby assign my copyright of this thesis to the Charles Stark Draper Laboratory, Incorporated, Cambridge, Massachusetts.



Jason R. Schott, 2Lt., USAF

Permission is hereby granted by the Charles Stark Draper Laboratory, Incorporated to the Massachusetts Institute of Technology to reproduce and to distribute copies of this thesis document in whole or in part.

This thesis is dedicated to my wife, Emily,
and to the child she carries for us.

Table of Contents

List of Figures

List of Tables

Nomenclature

γ, μ	repair (transition) rate
λ	failure (transition) rate
σ_{share}	equivalence class sharing niche radius
A	transition matrix
cfe	cost function evaluation
DM	decision maker
e	efficient set representation
E	actual problem efficient set
F	fitness
g	function constraint
G	multiplicative fitness penalty function
ga	genetic algorithm
J	cost functional
J	vector of cost functionals (multicriteria design)
l	number of criteria
l	binary string length
m	number of function constraints
MCGA	multicriteria genetic algorithm
n	number of design parameters
p	Holder metric degree
P	population size
P_{dom}	domination tournament fraction times population size
PBL	product of bit likeness convergence criterion
ssga	steady-state genetic algorithm
T_1	candidate domination tie defined as when only one candidate does not dominate the entire tournament set
T_2	candidate domination tie defined as when both candidates dominate the same fraction of the entire tournament set
t_{dom}	domination tournament size expressed as a fraction of the population size
tga	traditional genetic algorithm
x	design parameter
x	vector of design parameters

1.0 Introduction

1.1 Background

Fault tolerance is defined as the ability to continue operating after the failure of a given system component. To be fault tolerant, a system must have one or more redundant components that can take over the function when the primary component fails. In addition, the system must have both a means of detecting failures in the components and a means of transferring to working components after a failure has been detected. Fault-tolerant system configurations are used extensively in processes where the system must remain on-line in the event of component failure. Although applications are widespread, industrial processes, aerospace vehicles, and ground transportation are especially noteworthy. The need for optimization in the design of these systems is apparent when one thinks of the complexity of modern spacecraft. The high costs associated with their fabrication and launch dictate that any design proposal be assured a very high probability of success at the lowest possible system cost.

This thesis attempts to combine the robustness of the genetic algorithm with the proven effectiveness of the Markov Modeling method and the Optimal Design Process for fault tolerant system design. The efforts of this thesis are divided into two main sections: the first deals with the application of the genetic algorithm to single criterion fault tolerant design, while the second deals with the application of the genetic algorithm to multicriteria fault tolerant design.

Chapter 1 provides a brief overview of the Markov Modeling Method, the Optimal Design Process, and general mathematical programming. Chapter 2 details the fault tolerant test problems used throughout this research. The next chapter introduces the concepts involved in genetic algorithm optimization and is followed in Chapter 4 by discussion of the Steady-State Genetic Algorithm developed independently for this research. The next two chapters, 5 and 6, discuss the difficulties involved in handling functional constraints and determining convergence in genetic algorithms. Chapter 5 introduces a new concept called "fitness penalty" which uses the strengths of the genetic algorithm to create general, effective penalty functions. Chapter 6 provides an outline of genetic algorithm convergence and the difficulty in determining a termination criteria. A method of predicting convergence is presented that relies on a measure of diversity in the genetic algorithm. The results for mutation rate and population size analyses are presented in Chapters 7 and 8, respectively, followed by a summary of genetic algorithm single criterion optimization in Chapter 9.

The second section of the thesis begins in Chapter 10 with a background summary of multicriteria optimization and the various types of methods that fall into this category of mathematical programming. Chapter 11 illustrates two common generating techniques of multicriteria optimization that have applicability to fault tolerant design, followed by the introduction of a genetic algorithm that uses multicriteria dominance as its selection technique. This multicriteria genetic algorithm, or MCGA, is investigated in greater detail by the remaining chapters. Chapter 12 summarizes the parameters of the method that effect its performance and Chapter 13 lists some criteria by which its performance can be measured. The performance of the MCGA on two criteria fault tolerant system design problems is investigated in Chapter 14, including the effect of three of the performance parameters. Chapter 15 compares the performance of the MCGA to that of the constraint method, followed by a general summary of multicriteria optimization in Chapter 16. The final chapter of the thesis, Chapter 17, lists the various avenues of effort that further work could pursue to enhance the knowledge in this field.

Before the methods of optimization that will be investigated are presented, the reader must understand the framework of this analysis. For this, we begin with an introduction of the modeling process of fault tolerant system design.

1.2 The Modeling Process

The design of fault tolerant systems relies heavily on the availability, accuracy, and completeness of a model representing the system design. The development of an appropriate model becomes especially complex as component interdependencies and failure rates propagate with time. This section provides a basic outline of the modeling process for fault tolerant design.

The process of generating a reliability prediction for a system can be divided into three steps [1]. First, the system needs to be carefully examined. The goal is to discover how the system operates and what are its critical aspects. This step results in a system description. Second, the impact of failures is explored. This step is often called a failure modes and effects analysis (FMEA). During this step, the accident modes of the system are delineated. Third, the Markov model is constructed. Information on system operation from step one is used to guide modeling decisions such as the proper representation for the human elements (this reflects the manner in which personnel affect the system operation). The model is a systematic representation of the FMEA from step two.

The actual process of generating a model requires information on: architecture, component characteristics, operational requirements and reconfiguration procedures. The system architecture provides information such as what components exist and how they are connected, both physically and logically. The model also needs various component characteristics, such as failure and repair rates. The operational requirements provide a definition of what equipment or abilities are needed to achieve an operational state. The reconfiguration procedures are the actions taken when a failure occurs so that system operation remains in the most desirable mode.

The model indicates when certain operational decisions impact safety (such as the rate of restoration of a specific safety function following a component failure). Also, sensitivity analyses indicate how different modeling assumptions and uncertainties in model inputs affect the results. A complete description of the Markov Modeling Method can be found in Appendix A.

1.3 Optimal Design Process

Optimization implies the process of determining a set of function parameter values that cause a process to satisfy specified constraints and at the same time maximize (or minimize) some performance criteria (expressed as cost functions). Fault-tolerant system optimization normally requires extensive and usually subtle tradeoffs between factors such as component quality, reconfiguration strategies, level of redundancy, and operational policies. Optimization strategies must incorporate the conflicting effects of such constraints as performance specifications, reliability goals, and size and weight in order to design to minimize cost. The complexity of this process requires the availability of a systematic and efficient design approach capable of dealing with large numbers of components arranged in any number of ways.

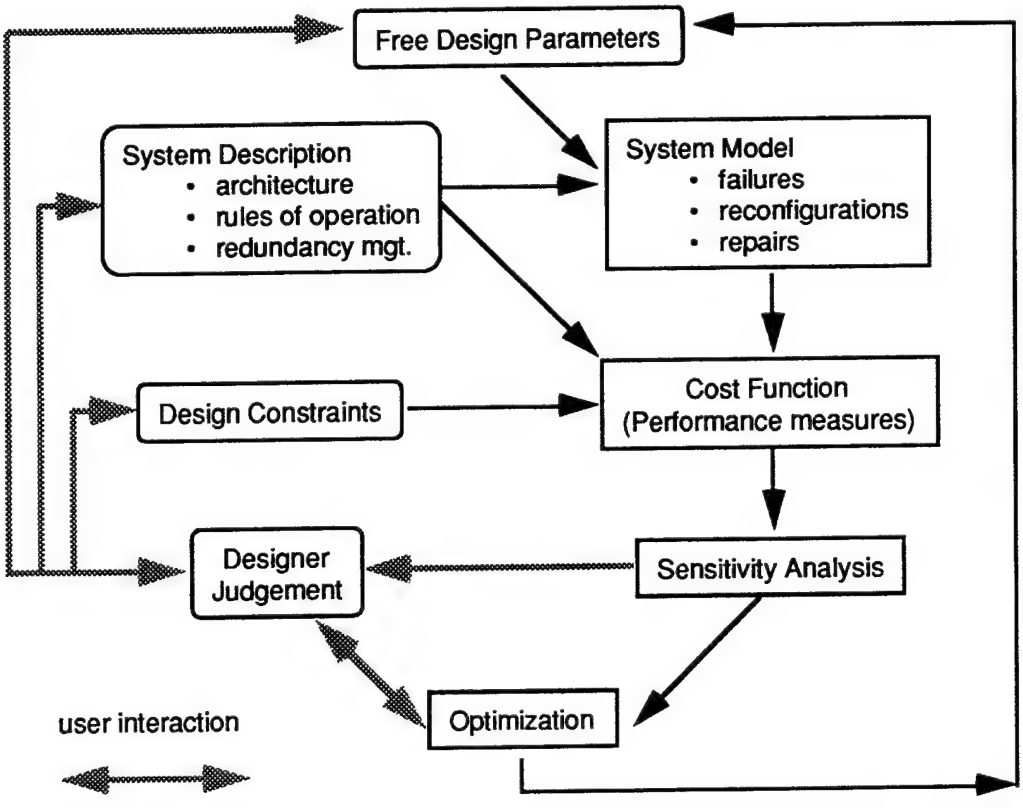


Figure 1-1: Optimal Design Process

Such a design approach should create a suitable system model and apply a computational algorithm that adapts readily to model changes and

reaches a satisfactory solution in a reasonable amount of time. As shown in Figure 1-1, the optimal design process can be formulated much like the optimal control problem. The solid arrows show the analytic path that correlates to the feed-forward path of control, with constant interaction from the external "control constraints" shown as the left column of boxes. Optimization forms the feedback portion of the control loop and completes the control-analogy to the optimal control problem. The behavior of the Optimal Design Process "plant" can be modeled to exhibit the same behaviors associated with control. This framework allows us to use much of the existing knowledge in optimal control to systematically solve the complex fault-tolerant design problem.

Markov modeling techniques (Appendix A) have been increasingly used for reliability prediction. These techniques in conjunction with the Optimal Design Process framework have also been used successfully to aid in the design of fault tolerant systems. Specifically, past research efforts at the Charles Stark Draper Laboratory have created a general framework for integrated system optimization incorporating Markov models in the Design Optimization/Markov Evaluation (DOME) program.

A Markov reliability model calculates the probability of the system being in various states as a function of time. A state in the model represents the system status with respect to component failures and the behavior of the system's redundancy management strategy. Transitions from one state to another occur at given transitions rates which reflect component failure and repair rates and redundancy management performance. Each element in the model's state vector represents the time-dependent probability of the system being in a specific state. Since the Markov model traces the evolution of state probabilities **P** based on the above mentioned transition rates, it is not explicitly simulating the system and therefore does not have the deficiencies associated with Monte Carlo techniques (see [1] for details on the Monte Carlo method as an additional reliability analysis technique). Sequence dependencies, such as repairs and redundancy management decisions, are included naturally. The Markov model is cast into a system of ordinary differential equations of the form:

$$\dot{\mathbf{P}} = \mathbf{A}(\alpha)\mathbf{P} \tag{1}$$

where **A** is the nxn transition matrix whose elements represent the transition

rates between system states, and α is the vector of input design parameters such as failure, repair or reconfiguration rates. As an example, a simple four state Markov system model is shown below.

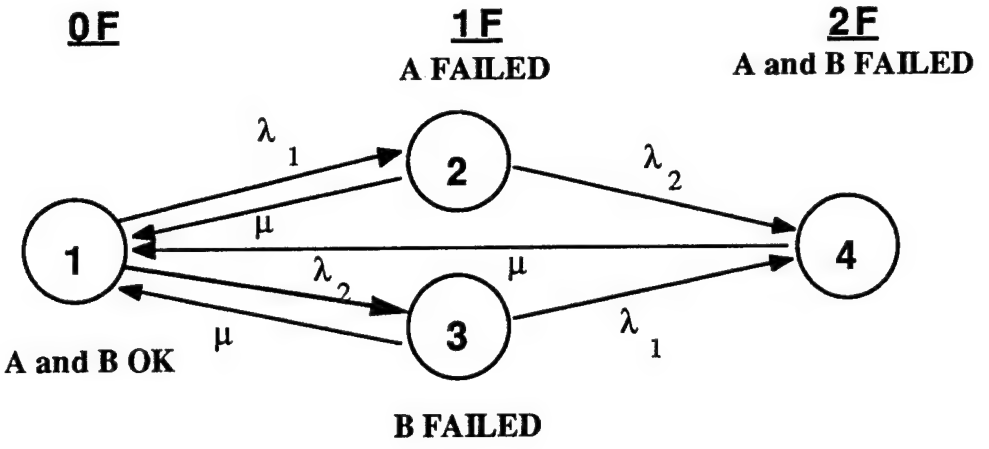


Figure 1-2: Simple Markov model for a dual component system

The decay of the system from state 1 to state 4 is denoted by the failure rates λ_1 and λ_2. Repairs are effected at the rate μ. This basic Markov framework can be expanded to model any fault tolerant system. A complete description of the process involved in obtaining Figure 1-2 is provided in Appendix A: "The Markov Modeling Process".

Furthermore, the differential nature of the model means that it is not necessary to generate *explicitly* all possible combinations of events that can occur over the entire time period in question; rather, it is only necessary to model events that can occur during an infinitesimal time step. Of course, there are also some drawbacks to this method. First, the state space grows exponentially with the number of components. Nevertheless, techniques have been developed to render this drawback tractable in many situations of interest. Reference [1] covers this dimension problem in greater detail for the interested reader. The second drawback is that treatment of complex mission scenarios and repair strategies, although possible, are generally cumbersome.

To summarize, the merging of the framework of the Optimal Design Process with the versatility, effectiveness, and efficiency of the genetic algorithm should provide a state-of-the-art capability for dealing with formerly intractable problems. It is the goal of this thesis to verify the validity of this assertion.

1.4 Single-criterion versus Multicriteria Optimization

As stated above, fault-tolerant design requires extensive tradeoffs between factors such as component quality, reconfiguration strategies, level of redundancy, and operational policies. Therefore, optimization strategies must somehow incorporate the conflicting effects of such criteria as performance specifications, reliability goals, and size and weight in order to design to minimize cost. The Optimal Design Process is a very effective means of dealing with these conflicting criteria and generating a realistic system model.

Whenever possible, the designer attempts to combine all criteria of interest into a single cost function. The creation of a single figure-of-merit for a problem enhances the ability of a search method to quickly and accurately generate a solution and often increases the decision maker (DM) satisfaction with the solution produced.

The three most common criteria in fault tolerant design are (1) procurement cost, (2) availability, and (3) operating cost. Procurement cost is a configuration fixed quantity. It is solely dependent on the initial conditions for the problem. The availability (reliability) of the design depends on the states of the Markov model at the terminal conditions. The operating cost, however, is not a fixed time quantity. Operating cost is determined as an integral-over-time condition over the system's life cycle.

A common example of this approach would be fleet design for an airline. Some airlines employ a "deferred maintenance" concept, which strives to meet specified goals but allows maximum maintenance flexibility to decide when an where maintenance should occur. The issues involved include keeping aircraft operational, reducing the locations where heavy maintenance is performed, reducing the heaving maintenance frequency, reducing emergency repairs, and others. A single cost function that could be created for this type of problem is shown in equation (2):

$$\text{Cost}_{\text{total}} = \text{Unavailability}_{\text{cost}} + \int_{0}^{\text{Lifecycle}} \{\text{Replacement}_{\text{costs}} + \text{Repair}_{\text{costs}} + \text{Waste}_{\text{costs}}\} dt \quad (2)$$

If a relationship such as this between procurement cost, availability, and operating cost is known or satisfactorily approximated, a single scalar function can be formulated. If, however, a cost cannot be readily assigned to

availability, a multicriteria problem must be solved.

Criteria are not always commensurate. For example, the design of a system may have to account for the desire to minimize collateral damage and mission preparation. The worth of human life and the value of time are incommensurate to most decision makers (DM) because of the difficulty involved in resolving the relative worth of human life in terms of hours. In such an instance, another framework for the design process is desired that takes conflicting criteria into account without unduly sacrificing the reliability of the solution or the time required to reach it. The focus of multiobjective programming is to provide the DM with a means of understanding the tradeoffs involved in the problem and to help him or her to identify a quality solution.

Fortunately, multicriteria optimization problems also fit into the framework of the Optimal Design Process. Except for the fact that a single optimum solution does not exist in the normal sense and multiple cost functionals are maintained, Markov modeling still provides an effective means of achieving time-dependent reliability analysis on problem criteria of interest. Commensurate criteria should be combined into a single cost functional whenever possible, to avoid dealing with the complexities of multicriteria optimization and to regain the greater assurances of optimum.

1.5 Mathematical Programming

In any given search space for optimization, there are two types of optima: global and local. The definition of optima also changes depending on whether the goal is the maximum or the minimum of some value. Fault tolerant design deals primarily with costs, which almost always are to be minimized. As such, all optimums in this thesis should be considered minimums unless otherwise noted.

A global minimum indicates the location in the search space with the very lowest cost function evaluation. A local minimum, on the other hand, represents the lowest cost function value over a limited portion of the search space. All optimization processes intend to locate the global minimum, the constraints (either external or internal) imposed on optimization in a complex problem keep the search algorithm from complete assurance of finding the global minimum. Additionally, local minima may exist in the design space in the complete absence of constraints. Consequently, successful optimization

hinges on the search algorithm's ability to locate a minimum that satisfies performance criteria.

The fault-tolerant problem formulation developed here must be solved by some combination of four mathematical programming approaches:

1) Linear programming where both objective and constraints are linear functions of decision variables,
2) Non-linear programming where at least one function is non-linear,
3) Integer programming where the solution must lie in the integer set,
4) Discrete programming where it is required that the solution lie within a chosen set of discrete values.

DOME has a versatile and well-tested continuous parameter optimization capability capable of solving non-linear programming problems [8]. This means, however, that like most capabilities available today, it lacks the means of dealing with many real-world problems. The ability to deal with integer and discrete problems are important because realistic design must often deal with a finite number of available components, quality levels, limited personnel expertise or scheduling conflicts, equipment settings, etc.

Dealing with discrete programming problems is not a trivial undertaking. George Dantzig, the creator of the famous simplex linear programming method, once said that the first phase in doing an integer programming problem is to try to convince the user that he or she does not wish to solve an integer programming problem at all! [5]

We have chosen to develop the capability to perform discrete programming into the DOME framework because unlike other analytic approaches that require explicit formulas for state probabilities, DOME is a general framework applicable to a wide range of real-world fault tolerant design problems. In addition, DOME's modular design readily incorporates new developments in optimization techniques—which makes it perfect for testing the benefits genetic algorithms can bring to discrete and mixed continuous/discrete fault tolerant design optimization.

2.0 Description of Test Problems

Four fault tolerant test problems have been selected, representing three stages of increasing complexity. The four are referred to as Warning Lamp, Asymmetric Lamp, Triplex, and TISS.

2.1 Warning Lamp Problem

The most basic problem used is the Warning Lamp problem described in detail in [8]. Briefly, Warning Lamp is a design problem for a dual redundant fault tolerant system. The system makes use of a "warning lamp" to signal a system operator that a manufacturing process requires adjustment to continue producing usable product. Failure to make the adjustment will result in production of unusable product and lost profits. At the end of the production run (system life cycle), the system will be subject to an audit and total failure of the system at this time will result in a cost penalty.

The design specifies that a dual redundant lamp will be used. Each lamp will be testable for malfunction and repairable only by qualified repair personnel, who check the system at regular intervals. The design goal is to maximize life cycle profit by choosing the optimal quality lamp for each of lamp 1 and lamp 2 and scheduling an optimal repair schedule. The better the quality, the more the lamps cost; adding repair visits increased cost; and emergency repair visits between scheduled intervals due to failure of both lamps requires high cost penalties.

Warning Lamp was used to verify the single criterion genetic algorithm (ga) code development and is shown here to illustrate the typical fault-tolerant

design formulation. It is also used as a continuous parameter problem to compare the ga against the present capabilities of DOME. The derivation of the Markov model for this system is described in Appendix A.

There are four Markov states in this three parameter model (see Figure 1-2) that correspond to both lamps operational, the loss of the first lamp, the loss of the second, or both of the dual redundant lamps off-line. The two lamps contained identical components and repair is allowed from all failure states. The search space of this problem has a single, well defined minimum with gentle characteristics in the vicinity of the minimum. The smoothness of this simple problem makes it especially easy for gradient based optimization methods. The figure shows the shape of the cost function when the failure rates are not allowed to vary independently.

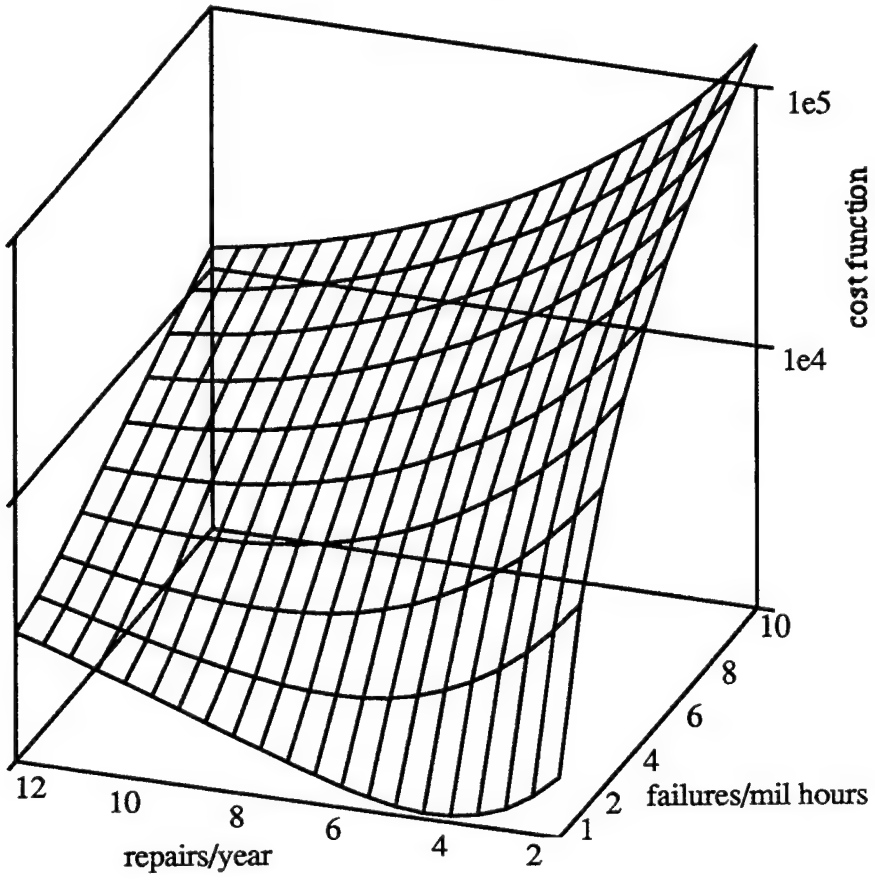


Figure 2-1: Mesh plot of Warning Lamp problem

The cost function for this problem is formulated as (see Appendix A):

$$J = P_4(T)C_{fa} + \int_0^T \{\gamma[C_r + (P_2C_{lp1}) + (P_3C_{lp2}) + P_4(C_{lp1} + C_{lp2})] + P_4C_{pl}\}dt \qquad (1)$$

The cost function has many components. First, regardless of whether a failure occurs or not, the repairman costs C_r for each visit. For this example C_r was \$25. The frequency of his visits (i.e. maintenance frequency) is a continuous design variable γ. The second, third and fourth terms in the integral are the cost of lamp replacement, C_{lp1} and C_{lp2}. The problem statement allows for the choice between three different, distinct lamp qualities:

lamp quality	MTBF (hr.)	cost per lamp
average	1e+05	\$ 1
high reliability	5e+05	\$ 10
ultra high reliability	10e+05	\$ 100

Table 2-1: Lamp cost versus quality level

Note that mean-time-between-failures (MTBF) for the average quality lamp is slightly higher than the 10 year (87,600 hour) life cycle. The cost of lamp quality in the cost function is approximated by a polynomial curve fit across the range of qualities. This polynomial curve fit is necessary for most optimization methods that directly apply or indirectly use continuous optimization techniques (see Appendix B: "The Branch and Bound Method") to arrive at the final discrete solution. The importance of curve fit is mostly eliminated here since the fit matches the actual cost at the discrete points of interest. The final term in the integral is the cost of operation when a dual failure exists. The profit loss rate C_{pl} was \$1000 for this example. The terminal portion of the cost function is concerned with conditions that exist at the end of the production run. If the system is found to be in the dual lamp failure state at the end of production, $P_4(T)$, a cost penalty of C_{fa} equal to \$100,000 will result, representing an entire lost batch of product.

The state transition matrix is

$$A = \begin{bmatrix} -(\lambda_a + \lambda_b) & \gamma & \gamma & \gamma \\ \lambda_a & -(\lambda_b + \gamma) & 0 & 0 \\ \lambda_b & 0 & -(\lambda_a + \gamma) & 0 \\ 0 & \lambda_b & \lambda_a & -\gamma \end{bmatrix} \qquad (2)$$

This state transition matrix A is representative of those of the remaining problems in the test set. Its development is detailed in Appendix A, where the Markov modeling process outlined. Note that each column of A adds

to zero, allowing Equation (2) to step forward in time via ordinary differential equations. This is only one of the reasons that the Markov modeling method is recommended for fault tolerant system design.

2.2 Asymmetric Lamp Problem

The second stage of testing consists of two problems of higher complexity than that of the Warning Lamp problem. The first is called the Asymmetric Lamp problem and is derived from Warning Lamp.

In the Warning Lamp problem, a manufacturing system makes use of a dual warning lamp to signal an operator that a manufacturing process requires adjustment to continue producing a usable product. Once a dual lamp failure occurs (both lamps failed simultaneously), it is assumed that the system immediately needs adjustment and produces an unusable product, and loss of profits, until the lamps are repaired and the process adjusted. To insure proper working of the lamps a qualified repairman is contracted to check the system on a regular interval, ranging from as frequent as once every three days (87.72 hour intervals) to as infrequent as not at all during the ten year life cycle (10,000 hour intervals). Each visit requires a repair fee and the cost of any lamps replaced. Moreover, at the end of the production run, the system is subject to an audit and total failure of the system at this time results in a cost penalty. The three design variables in the problem were the two lamp qualities and the repair frequency.

The primary difference between the Warning Lamp problem and the Asymmetric Lamp problem is that the lamp failures will be considered discrete variables instead of continuous variables. In this way, we more accurately model the actual system. Additionally, we will know that the solution obtained is indeed an extremum for the situation considered. We will not have to concern ourselves with rounding to the nearest lamp quality level, as a solution to the Warning Lamp problem may require. Also, we will know that the cost (problem solution) will be accurate since we will not have to interpolate between costs for different quality levels. The repair interval remains continuous since the repairman can come as frequently or infrequently as desired. The resulting problem has much of the same characteristics as the Warning Lamp, but it represents a mixed continuous and discrete parameter problem more typical of fault tolerant design.

To observe optimization methods' performance more clearly and increase the problem complexity, the cost of one of the lamps is altered to create an asymmetric solution. One of the lamps is considered the primary lamp that must put out more light and so costs more. The actual qualities and costs are given in the table below.

lamp quality	MTBF (hr.)	cost per lamp A	cost per lamp B
average	1e+05	$ 1	$ 100
high reliability	5e+05	$ 10	$ 1,000
ultra high reliability	10e+05	$ 100	$ 10,000

Table 2-2: Asymmetric lamp cost versus quality level

Other than the change in lamp cost, the rest of the problem is similar. Both the Asymmetric Lamp and Warning Lamp problem can be represented by the diagram in Figure A-3, the Markov model in Figure 1-2, and the cost function in equation (1), shown below. As Table 2-2 shows, the Asymmetric Lamp has the same failure rates for both lamps, but the two lamps are treated as different cost components in the cost function. In other words, while both lamps choose between components of the same quality range, the components for lamp 2 cost two orders of magnitude more than those for lamp 1. This creates the problem asymmetry that generates a search space with multiple optima. Equation (1) is shown to help illustrate the form of the cost function:

$$J = P_4(T)C_{fa} + \int_0^T \left\{\gamma\left[C_r + (P_2C_{lp1}) + (P_3C_{lp2}) + P_4(C_{lp1} + C_{lp2})\right] + P_4C_{pl}\right\}dt \qquad (1)$$

The cost function has many components. First, regardless of whether a failure occurs or not, the repairman costs C_r ($25) for each visit. The frequency of the visits is a continuous design variable γ. The second, third and fourth terms in the integral are the cost of lamp replacement, C_{lp1} and C_{lp2}. The final term in the integral is the cost of operation when a dual failure exists. The profit loss rate C_{pl} was $1000 for this example. The terminal portion of the cost function is concerned with conditions that exist at the end of the pilot production run. If the system is found to be in the dual lamp failure state at the end of production $P_4(T)$, a cost penalty of C_{fa} equal to $100,000 results. The state transition matrix (A) is the same as in equation (2).

This problem is used to test the genetic algorithm's capability to optimize problems with both continuous and discrete parameters. In order to do this, the two failure rates are treated as discrete parameters with 16 bins (discrete values) for each, while the repair rate is kept continuous (represented by 255 discrete points by the genetic algorithm methods). The remaining problem information can be found in Table 2-3.

String length	16
Continuous parameters	1
Discrete parameters	2
Values per discrete parameter	16
Design space points	65280
Markov model states	4
Model time duration	10 years

Table 2-3: Asymmetric Lamp problem data

There are three known minima found by exhaustive search of the design space that are used to provide a reliability comparison of the discrete optimization methods used in this thesis. These minima are described in Table 2-4. Parameters one and two are the discrete failure rates and the third is the continuous repair rate. The parameter values are given in a normalized range from 1.0 (their highest value) to 0.0 (their lowest).

	cost	parameter		
		(1)	(2)	(3)
1	0.0	1.0	0.5	0.72
2	185.3	1.0	0.0	0.77
3	254.6	1.0	1.0	0.00

Table 2-4: Asymmetric Lamp problem minima

The costs of Table 2-4 are also normalized. A value of 0.0 represents the global optimum, while 100 is the value Monte Carlo analysis produced for 400 points.

2.3 Triplex Problem

The other problem of the second level of problem complexity is the TRIPLEX problem. Triplex is a system containing three redundant components, with assumed operational capability when at least one component is still functional. The parameters of interest are the failure rates for the three system components. Repair is kept as a fixed parameter not used in optimization to limit the problem complexity. It is initiated following the second failure or at regularly scheduled intervals of $\mu = 0.2$ (5 hours). Each of the failure rates is allowed to vary independently.

Triplex is treated as the first all-discrete problem. It contains 8 states in its Markov model:

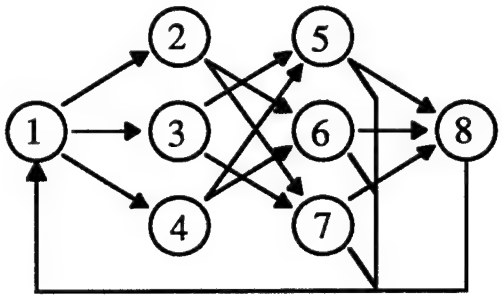


Figure 2-2: Markov model of TRIPLEX problem

As in the previous two problems, the cost functional contains an integral portion, reflecting the cost of repairs over the system life cycle, and a terminal part, representing a penalty for the system not being available at the end of the scenario. The resulting cost functional for the TRIPLEX problem is represented as:

$$J = \left[\frac{1}{\sum_{j=1}^{4} P_j(T)} - 1 \right] a_1 + \int_0^T \sum_{i=5}^{8} \left\{ \left[\frac{a_2}{\mu} + \left(a_3 + a_4\mu + \frac{a_5}{\lambda^2} \right) \right] P_i(t)\mu \right\} dt \qquad (3)$$

In this equation, λ_k's are the component failure rates affecting P_i, μ is the fixed repair rate, and the a's are the cost factors. The state probabilities P_1 to P_4 are availability states, while P_5 to P_8 are unavailability (repair) states. The reciprocal of the availability states is used to determine the probability of

system unavailability at the end of the life cycle (T) of 1,000 hours. The failure rates are treated as discrete parameters with ten possible rates equally spaced logarithmically between 1e-05 and 0.1. The values shown in Table 2-5 are based on a replacement system cost of $2.5 million and represent the values needed for equation (3).

symbol	value	function
μ	2.0e-01	repair rate (hr^{-1})
a_1	2.5e+06	system replacement cost ($)
a_2	2.5e+04	overhead ($)
a_3	2.5e+01	labor rate ($/hr)
a_4	2.5e+04	repairability ($ hr)
a_5	5.0e+04	quality factor ($/$hr^2$)

Table 2-5: TRIPLEX cost function constant values

The cost factors (a_i) for the problem are chosen to bias the repair cost in the overall cost function, resulting in multiple optima in the feasible design space. The bias was created by multiplying a_1 by 0.01 to significantly reduce its effect on the cost function.

Design and implementation information for this problem is included in Table 2-6.

String length	12
Continuous parameters	0
Discrete parameters	3
Values per discrete parameter	10
Design space points	1000
Markov model states	8
Model time duration	1000 hours

Table 2-6: TRIPLEX problem data

Each of the parameters are given 10 bins. Providing ten bins causes an additional complexity in that 6 (16-10) bit combinations of each genetic algorithm string are simply not allowed. The treatment of these disallowed values is explained in Section 3.3.

TRIPLEX has five (5) known minima described in Table 2-7, found by an exhaustive search of the design space. The values are normalized in the same manner as was done in Table 2-4.

	cost	parameter (1)	(2)	(3)
1	0.0	0.51	0.51	0.51
2	493.6	0.99	0.49	0.49
3	493.6	0.49	0.99	0.49
4	493.6	0.49	0.49	0.99
5	3322.0	0.00	0.00	0.00

Table 2-7: TRIPLEX minima

Triplex is not a very complex problem, but the various unconstrained minima allow proper testing of the optimization methods' solution reliability.

2.4 TISS Problem

The final problem is the most complex by far and represents a real-world fault tolerant design optimization problem. TISS stands for Trans-Ionospheric Sensing System and is a problem actually analyzed for the US Air Force by C.S. Draper Laboratory, Cambridge MA. The TISS problem used here is a slightly simplified version consisting of a 33 state Markov model with 17 optimization parameters. The 17 parameters of the model represent the 15 major system components and two repair rates. Each parameter is provided 5 discrete component options of varying quality and cost.

The TISS configuration consists of a dual redundant system that will process and store Global Positioning System (GPS) data. TISS is designed to operate autonomously in remote regions of the world. Infrequent scheduled maintenance and critical unscheduled repair visits will be design variables with a high associated cost due to the remote location. Remote access to the system is permitted via three dial-up telephone lines. The design architecture will look at minimizing the total life-cycle costs over a 5.7 year period while considering such items as reliability, maintenance, overhead, and component costs.

Table 2-8 shows the implementation data for this problem.

String length	51
Continuous parameters	0
Discrete parameters	17
Values per discrete parameter	5
Design space points	7.63E+11
Global minimum	unknown
Markov model states	33
Model time duration	5.7 years

Table 2-8: TISS problem data

This problem is used for much of the analysis performed in this thesis. The Markov model is constructed following the guidelines of Appendix A, with some additional state reduction schemes employed to keep the problem tractable. The size of the resulting system model, with 33 states and two independent repair rates represents a practical example of realistic fault tolerant design. The computational effort needed to run this problem on several machines is shown in Table 2-9.

Platform	seconds per cost function evaluation
MacIntosh IIcx	10.5
MacIntosh Quadra 700	1.83
MacIntosh Quadra 950	1.55
IBM RS6000 Workstation	0.83
Quadra 700 with PowerMac Upgrade Card*	0.20 (* optimized compiler)

Table 2-9: TISS problem computation time on various platforms

Additional information necessary for the implementation of this problem can be found in [1].

3.0 Genetic Algorithms

3.1 Background

The genetic algorithm (ga) was developed a quarter century ago by John Holland, et. al., at the University of Michigan with two goals in mind: (1) to abstract and rigorously explain the adaptive processes of natural systems, and (2) to design artificial systems that retain the important mechanisms of natural systems [7]. Over the years, the genetic algorithm has developed into a reliable means of dealing with a wide range of problems. Our intent here is to maintain a high reliance on natural, evolutionary mechanisms to find quality solutions to difficult fault tolerant system design problems.

Genetic algorithms use random choice in a directed search process for optimization. This search is randomized, but its use of mechanisms found in natural genetics to improve the solution distinguishes it from random search methods in the strictest sense. The basic features of the ga that separate it from other methods are that it:

1) works with a coding of system parameters, not the parameters themselves,
2) searches from a population of points, not a single point,
3) uses payoff information only from the objective function, not derivatives or other auxiliary knowledge, and
4) uses probabilistic transition rules, not deterministic rules.

Genetic algorithms use *reproduction*, *crossover*, and *mutation* as their three basic operators. In the implementation of this thesis, the ga takes the form shown in Figure 3-1:

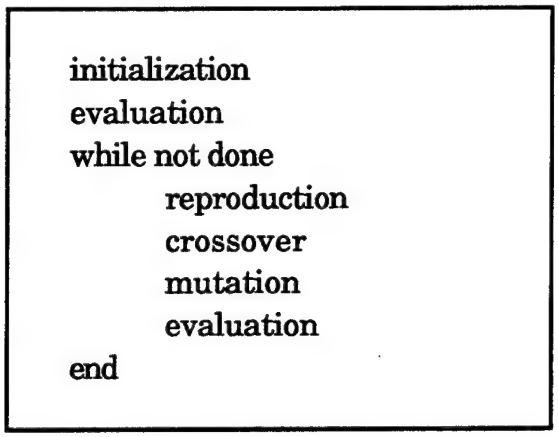


Figure 3-1: Flow diagram of the genetic algorithm

The ga works because it reproduces high-quality (fit) notions according to their performance. These notions are allowed to mate with many other high-quality notions of other strings to combine the notions into superior individuals. Finally, the crossover and mutation operators speculate on new ideas constructed at random (mutation) and from the experience of past trials (crossover). Mutation is a necessary operator because reproduction and crossover may lose some potentially useful genetic material; injecting diversity by way of mutations improves the probability of locating the global optimum. However, just as in natural genetics, it needs to be used sparingly to keep from disrupting beneficial information already in the population and to allow the algorithm to converge to a solution without unnecessary delay. Appropriate use of reproduction, crossover, and mutation enhance the *robustness* of the genetic algorithm as applied to the design of fault tolerant systems.

The notion of robustness is very important to the development and use of the genetic algorithm and can be described as follows. Suppose that the problem of interest P belongs to a set $\mathscr{P}$ of problems that contain all variations of P that meet some criteria of similarity. In this thesis, $\mathscr{P}$ contains all fault tolerant design problems with characteristics in common with those described in Chapter 2, "Description of Test Problems". A design method, M, is *robust* with respect to the optimization of P if its characteristic optimization performance, C, holds for every problem in $\mathscr{P}$. Robustness requires a set $\mathscr{P}$, a method M, and some characteristic C of M. In this thesis, robustness refers to the optimization of the problems described in Chapter 2, and the degree of *exploration* and *efficiency* that a method attains.

Exploration as defined here refers to an ability to master the design space—to provide outstanding solutions to $\mathscr{P}$ with incredible reliability. Exploration can also be described as the effectiveness of a method (efficacy). Perfect method exploration would achieve the global optima of every problem in $\mathscr{P}$ 100 percent of the time. Methods that achieve high effectiveness usually examine a large fraction of the design space. Therefore, exploration usually comes at a cost to *efficiency*.

Efficiency refers to the degree of effort necessary to achieve effectiveness. Efficient optimization methods spend little or no time searching non-optimal/locally optimal portions of the design space and proceed directly to global optimum.

In essence, robustness provides some balance between exploration and efficiency, allowing the search method's survival in many design environments. Robustness is desired because it reduces design effort and time and increases the quality of solutions produced. As stated in [7], "where robust performance is desired (and where is it not?), nature does it better; the secrets of adaptation and survival are best learned from the careful study of biological example".

Genetic algorithms are the product of such biological study. Twenty years of research has verified the robustness of the ga in general, and it is the intent of this thesis to prove their robust application to the design of fault tolerant systems.

3.2 Traditional Genetic Algorithm

The most common implementation of genetic algorithms is commonly called the traditional genetic algorithm (tga). In the tga, a *generation* represents the genetic algorithm population of members at the current "time". "Time" within the algorithm represents the linear progression of a population through successive generations and is analogous to time in an evolutionary sense. The population size (P) normally remains fixed from generation to generation. The genetic algorithm evolves its population in "time", always striving to improve the overall fitness ("worth" of individual members) of each successive generation.

The traditional genetic algorithm used in this thesis is a modified version of the classical tga used in single criterion optimization. It can be classified as a tga by having the following characteristics:

1) binary (0 and 1) parameter encoding
2) generational reproduction
3) fitness normalization
4) elitism (optional for tga classification)
5) crossover and mutation operators

This tga is modified only in characteristic 5 in that a two-point crossover is used instead of the traditional one-point crossover technique.

The tga developed for this thesis uses appropriate implementations of the three basic operators based on their ability to enhance the robustness of the method. The corresponding probabilities (settings) of the operators and the appropriate population size are examined and the results compared with other direct search methods. The aim is to establish robust values of these parameters applicable to many fault tolerant system design problems.

The diagram below shows how the tga operators manipulate population members in an effort to evolve a new generation, separate from the previous, that has a better average fitness.

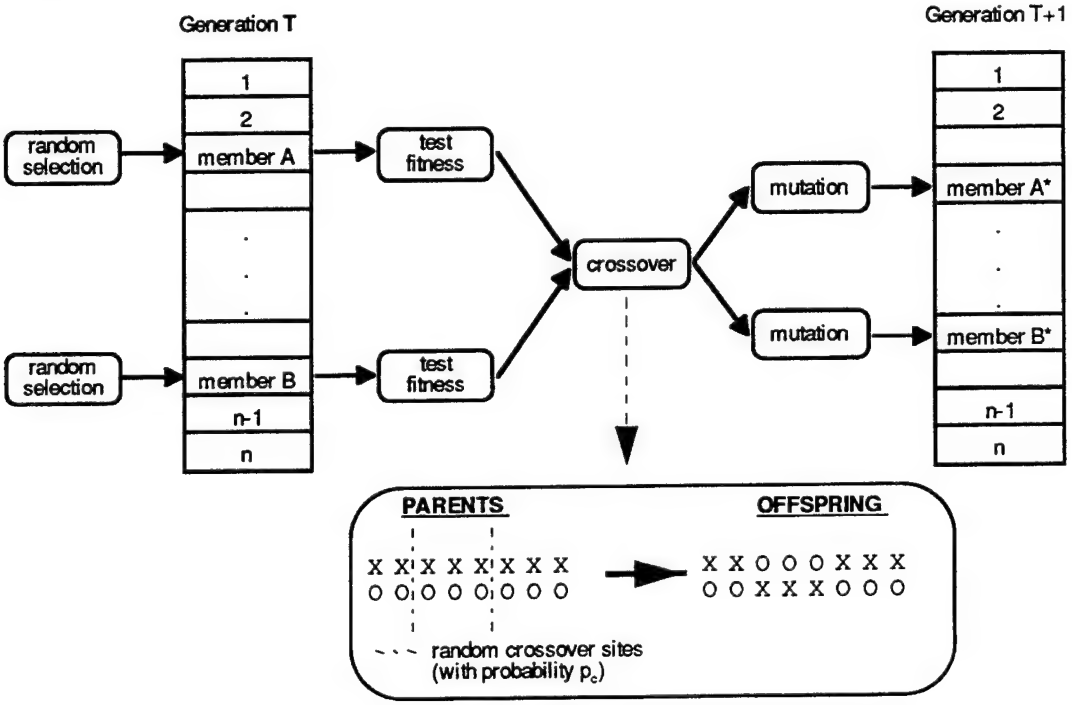


Figure 3-2: Traditional genetic algorithm reproduction cycle

3.3 Coding and Schema Theorem

Coding a problem into a form compatible with the ga is not generally difficult. The robustness of the ga allows it to be forgiving of the form the problem is presented in, but [7] provides two simple principles that reasonably govern the efficiency of the method:

1) The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.
2) The user should select the smallest alphabet that permits a natural expression of the problem.

To understand these principles, first one must know the "schema theorem." Schema theorem states that the ga works because it propagates encoded building blocks of valuable problem information at exponentially higher rates and in parallel through the use of a population.

Genetic algorithms exploit similarities in the string coding when the schemata (plural for schema) represent optima. A schema describes a template of similarity among ga strings. In the case of binary coding, the alphabet of the schema has three characters: 0,1 and *, where the "don't care" (*) signifies that we "don't care" if the bit in question is a 1 or 0. For example, the schema 0**1 on a four bit string can represent four different strings: [0001; 0011; 0101; 0111]. The length of a schema is the inclusive number of bits from its first to its last 1 or 0 bits. In schema 0**1 the schema length is 4, while *11****** has a length of two. Note that the * is only used for notation and is never explicitly referenced by the ga.

The schemata must be naturally represented by the alphabet used to allow the ga operators to fully exploit the design space. For example, consider how one could encode the first eight integers. A binary string of length 3 would fully represent these values, with one corresponding to [0 0 0] and eight to [1 1 1]. Another alphabet for this problem would be A-H, where one corresponds to A, two corresponds to B, etc.

In fault-tolerant design, like most discrete engineering problems, the parameters being coded are physical quantities (weight, size) or probabilistic quantities (failure rate, repair rate, coverage), which are naturally represented by either of the two forms given above. In the example above, both alphabets give a complete and accurate representation of the first eight integers.

However, what happens if only the first 5 integers are desired?

In this case, the second nonbinary alphabet can be reduced to A-E, to again represent the parameter. However, the binary alphabet is restricted to 2^l dimensions, where l is the length of the string. A binary representation of the first 5 integers requires the same binary string length as the first 8 (l=3). This leads us to have to consider what influence the extra three values will have on the ga operation, but first let us look at the impact of principle (1).

Principle (1) implies that the problem coding should provide a maximum of schemata. Alphabet cardinality influences the string length required to represent a parameter; to equate schemata of coding schemes of different cardinality, we can watch the relationship of binary string cardinality (2^l) where l is the binary string length, to nonbinary string cardinality represented as $k^{l'}$, where l' is the nonbinary string length:

$$\begin{aligned} \text{Binary string schemata} &= 3^l \\ \text{Nonbinary string schemata} &= (k+1)^{l'} \end{aligned} \qquad (1)$$

From these relationships, the number of schemata formed from a particular problem coding can be determined. Using the first 8 integers for example, k=8, l=3, and l'=1, such that the number of schemata are 27 for the binary string and 9 for the nonbinary string. Binary coding allows the maximum number of schemata to be available for ga exploitation.

Binary coding is desirable for design space exploration, but again we return to the impact of undesired bit combinations on a binary string. One means of restricting their impact is to only generate initial population strings in the feasible range, but the impact of mutation and crossover still applies.

This issue could be dealt with in one of several ways:

1) Fill the empty portion of the 2^l string with values from the acceptable set (i.e. [1,2,3,4,5,+ *1,2,3*] = 2^3 = 8).
2) Ignore the generated disallowed string(s) and retain the parent(s).
3) Reinitialize disallowed parameter values at random into the acceptable set.
4) Fix the higher order bit and/or place at the parameter limit (i.e. if 6, 7, or 8 are created, place the parameter at 5.
5) Revert the disallowed parameter to the acceptable value it possessed prior to crossover or mutation. All changes made to other parameters of the string are retained.

Option 1 places the repeated portions of the design space in a more favorable likelihood of occurrence unrelated to their fitness and therefore biases the optimization process. Obviously it is desirable to retain as much of the reproduction effort as possible, so options 2, 3 and 4 would negate much of the effort generated. Option 3 introduces a whole new, and highly schemata destructive operator into the algorithm, while option 4 negates the whole randomized notion of the ga and imparts a rigidity to the algorithm that ignores any beneficial information the parameter previously may have contained. Consequently, option 5 is used in this analysis. If the two-point crossover affects two or more adjacent parameters, only that parameter that becomes disallowed is reverted to the previous value to maintain as much of the reproduction effect as possible. This issue will be examined in Section 8.5 to observe its impact.

3.3.1 Fault Tolerant Parameter Coding

In this research the selection of a parameter value is limited to the number of discrete bins provided by the user, all with equal likelihood. The discrete bins are the acceptable solution values recognized by the decision maker (DM) for the problem. Some problems are not solely discrete however, and have a combination of discrete and continuous parameters. In such a case, 256 (eight bit representation) linearly spaced discrete values are created to represent each continuous parameter. Note however, that these strings are cast independently of what bin *values* correspond to the bin *numbers* the ga operates on.

For example, a fully continuous three parameter problem with parameter ranges: [1e-10 to 1e-04], [π to 2π], and [1+j10 to 1+j70], could be represented by a string:

$$[0\ 0\ 0\ 1\ 0\ 0\ 0\ 0|1\ 0\ 1\ 0\ 0\ 0\ 0\ 1|0\ 1\ 1\ 1\ 1\ 1\ 1\ 1] = [16|161|127] \qquad (2)$$

The ga does not care directly about the parameter ranges. The ranges are extraneous information used solely to determine string fitness.

The bit size of each parameter (word) on a string is calculated independently as well. The size of a word corresponding to a continuous parameter is 8 bits (0 to 255), while discrete parameter word sizes are calculated independently as a "best fit". For example, 3 bins fit into a 2 bit

word, 8 bins into 3 bits, and 9 bins into 4 bits. How the "wasted" space on the fourth bit in this last case (4 bits holds 16 numbers, but only 9 are used) affects the ga is explored in Section 8.5.

Continuous parameter optimization must ensure that the encoding resolution is fine enough to realize a continuous-like representation along the applicable axes. In order to keep the manipulations required by the algorithm as simple as possible for memory and speed considerations, all continuous parameter encoding in this thesis is done on single byte binary strings. A single byte, 8 bit, string contains integer values from 0 to 255. This choice was made with the full knowledge that it places a limitation on the algorithm, but that if a particular representation does not provide adequate resolution, it can be fixed by reducing the range of interest, using a longer string, or using the scaling technique described below.

In this research, using 256 values for continuous parameters may limit the sensitivity of parameters with large ranges. For example, if linear mapping is used for a parameter with the range [1e-10 to 1e-4], it would have bin values separated by 3.92e-7. In other words, the first three recognized continuous values would be 1e-10, 3.92e-7, and 7.84e-7. This type of mapping makes it unlikely for the algorithm to find minima located at the lower end of the range because the parameter values of the population will likely be evenly distributed across the entire range, making 1e-10 an unlikely possibility in normal population sizes. It is also impossible for the ga to locate any minima that may occur between 1e-10 and 1e-7, for example.

In the design of fault tolerant systems, the engineer is often looking at parameter values covering orders of magnitude, especially when considering failure rates and possibly even repair rates. In light of this, a *logarithmic mapping* is used in this thesis so that equal logarithmic spacing is provided to continuous parameters for better sensitivity at the lower end of parameter ranges. If the engineer believes that linear mapping is more appropriate to his or her problem, a simple change of the ga code will allow that capability.

If we wish to represent a parameter in the range [1e-10 to 1e-4] with an eight-bit string, we need to know how much resolution the encoding provides. Table 3-1 shows the associated resolution.

bit value	parameter value	difference
0	1.000e-10	
1	1.056e-10	5.57e-12
2	1.114e-10	5.88e-12
3	1.176e-10	6.20e-12
...	...	...
253	8.973e-5	4.73e-6
254	9.947e-5	9.74e-6
255	10.000e-5	10.53e-6

Table 3-1: Logarithmic parameter-to-string mapping

The worst resolution always occurs at the upper bound of the parameter when scaling is logarithmic, but in this case it never exceeds 11 percent of the value to which it corresponds. Therefore, the single byte continuous parameter representation should provide resolution sufficient to represent all but the most unusual parameter ranges.

3.4 Population Initialization

The first step of the ga is to create independent strings which form the initial population. The initial population of strings for the ga can be created in many different ways as long as a wide diversity of characteristics are represented, i.e. the first generation initialization should cover the search space as thoroughly as possible. In this research, the initial population is created at random by randomly generating parameter values on each string.

3.4.1 Initial Guesses

Sometimes the decision maker (DM) has a good idea where the parameter values should lie and wishes to inject that hypothesis into optimization routine. This initial guess is mandatory in optimization tools that do not operate on a population. Such methods are also *extremely* sensitive to the quality of the guess unless a means of adding robustness, such as simulated annealing, is applied. Regardless of the safeguards used, however, the ga has the vast advantage in that instead of requesting a *good* guess, it

requests *diverse guesses*. As a consequence, the ga exploits the advantageous schemata of a DM provided guess and ignores the rest of that string.

3.5 Reproduction

In this thesis, reproduction occurs in one of two ways, depending on whether single or multicriteria optimization is performed. Single criterion ga reproduction *selection* and *fitness testing* to determine which individuals of the population will be reproduced based on factors equivalent with environment, mating preferences, and individual strengths. The second manner of reproduction, used in multicriteria optimization, incorporates *tournament selection*, and is described in Section 11.3.1.

Selection is simply the means by which members are "selected" for fitness testing. Fitness scaling maps the relative "worth" of strings of the population into a simple, strictly positive function that the ga can easily recognize and optimize. It has the additional benefits of separating the ga optimization from the underlying problem complexities, which adds to the method's versatility, and allows the incorporation of "survival of the fittest" into the optimization procedure via a standardized, normalized, and readily interpretable scale of relative string worth. Adapting the fitness coefficients to penalize members that violate function constraints is another benefit of fitness mapping (see Chapter 5).

3.5.1 Selection

Selection is performed randomly to enhance the diversity of the mating pool. Two members of the current population are chosen to be parental strings. Parents that "survive" have the ability to mate and produce offspring that (1) form a new generation in the case of the traditional ga or (2) strengthen the population in the case of the steady-state genetic algorithm (defined in Chapter 4).

First, two members of the current population are chosen at random to reproduce. In order to actually reproduce, however, these members must meet the current fitness requirement. This is accomplished by the algorithm by comparing the evaluated fitness of each member to a random number—if the fitness exceeds that number, the member is allowed to reproduce, otherwise

another member is selected at random and is tested in the same way until two members have passed the test. In this way, all members have a chance to reproduce, but the most fit members are more likely to pass the fitness test and will consequently produce more offspring. Hence, survival of the fittest!

3.5.2 Fitness Scaling

Fitness is a criterion for parent "survival". Fitness testing corresponds to the Darwinian theory of "Survival of the Fittest". A fitness function, which provides a mapping of the problem cost function on a scale of string "worth" or "value" recognizable by the ga, is formulated as a scale against which strings can be compared to determine their expected ability to improve the population.

Optimization is normally a problem of maximizing some performance criterion, and the genetic algorithm is no exception. Fault tolerant system design as it is addressed here (as well as many problems in optimization), however, is one of minimization, where the choice and configuration of parameters are meant to minimize some cost function—a lower cost represents a better system. One of the benefits of the genetic algorithm is that it doesn't care what the problem looks like, nor does it care what kind of cost function is involved. All that matters to the algorithm is that it has a string (chromosome) that it wishes to manipulate in order to *maximize* some fitness function.

The fitness function and the cost function must have a one-to-one relationship, but they can be arbitrarily related. In fact, the simplest way to make the genetic algorithm minimize a cost function is to make the fitness function that the algorithm is concerned with inversely related to the cost function. By making high fitness evaluations equal to low costs, the algorithm minimizes without even knowing it.

The fitness scaling function should be created to maintain suitable disparity between all members of the current population. This disparity allows all members the opportunity to generate offspring while the scaling keeps the proportion of the worst members only high enough to maintain population diversity. This diversity is a key advantage of the genetic algorithm because even though a particular string may have a high cost, it may contain schemata particularly beneficial to reaching a global optimum when combined with the schemata of other strings through crossover.

The most common methods of fitness mapping are cited in [7] and [3]. *Roulette wheel* fitness provides fitness based on a string's cost relative to the rest of the population. *Windowing* uses the costs associated with the string, and merely adds or subtracts a limit to influence the fitness values. Finally, *linear normalization* orders the strings from a fixed maximum at some decrement. The first two are particularly oriented toward maximization problems, which are not of interest in the context of most fault tolerant design. Linear normalization, or simple ordering, is an effective methods that accomplishes minimization as well as maximization by producing a partial order of strings. It has the drawback, however, of requiring the entire population to be sorted for each new individual. None of these methods warrant use in serious fault tolerant system design optimization.

One of the most powerful and simplest fitness schemes is *inverted linear fitness scaling*. It accomplishes the goal of string competition because it provides a complete mapping from cost function to fitness and it has excellent qualities for function handling. Fitness normalization, function inversion, and strictly positive fitness are all easily created using linear scaling for superior fitness mapping results. As noted by [7], linear scaling helps prevent the early domination of extraordinary individuals, while later encouraging a competition among similar strings. In the tga, the fitness scale is updated each new generation when the new population is ready to begin mating.

The following method of fitness scaling, suggested by Dr. W.E. Vander Velde of MIT, begins by making a linear fitness function with a negative slope. The scaled fitness function for the entire population is:

$$\frac{F_i - F_{max}}{F_{min} - F_{max}} = \frac{J_i - J_{min}}{J_{max} - J_{min}} \tag{3}$$

F_i represents the fitness value of the current string under consideration with associated cost J_i. J_{max} and J_{min} are the worst and best cost function values for the current population of strings. The maximum fitness value F_{max} for this method is 1.0, and the minimum F_{min} is 0.05. The F_i denote probabilities of mating. Therefore, the lowest probability for reproduction of any string in a generation is fixed at 0.05. On the other hand, the most fit member of a generation is always wanted for reproduction, so its fitness value of 1.0 assures certainty of reproduction. This fitness-to-cost function scaling is illustrated in Figure 3-3.

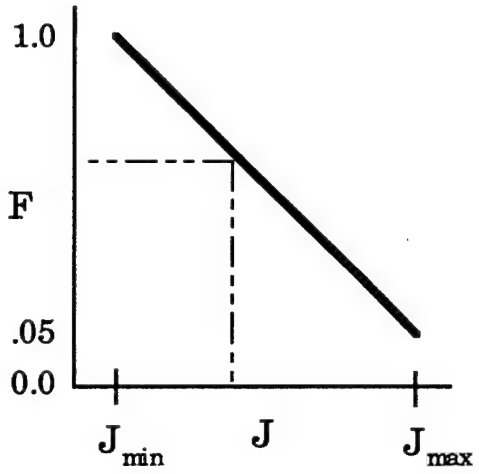


Figure 3-3: Inverted linear fitness scaling

These bounds were chosen so that the lowest cost members of the population will always mate if they are chosen by the random operator of the selection step, while the highest cost members will be allowed some, if very small, chance of reproducing. This scaling method is very robust in that it doesn't care what the associated cost function values are—they can be positive or negative or both.

In all of the design problems of this thesis, the objective of the decision maker (DM) is to locate the single, best optimum of the problem. In some design problems, however, the DM may have several regions of interest that she wishes to examine or she may wish alternative solutions to be created in addition to the overall optimum. In either case, *normal fitness sharing* can be applied to spread the ga population among the optima of the search space, with each optimum getting a fraction of the population proportional to its relative fitness. Fitness sharing is discussed in Section 11.3.1 for the interested reader.

3.6 Crossover

Crossover mates parental strings two at a time. Crossover occurs at a rate (probability) of p_C to capture and combine the beneficial traits of the different members. There are three main types of crossover in use today: (1) single-point crossover, (2) two-point crossover, and (3) multi-point crossover.

Single-point crossover simply chooses a random "cut" site on the string pair and causes the strings to exchange all bits on one side of the cut. This crossover propagates schemata of short defining length at a greater rate. These short schemata are reproduced at an exponentially higher rate. This procedure is called *Implicit Parallelism*, and is the fundamental feature behind genetic algorithm success.

Implicit parallelism, in short, is the simultaneous allocation of search effort to many "ideas", or hyperplanes, in the search space. Different strings of the population look at the merits of the different parameter values. This concept holds a diverse set of parameter combinations separately and propagates only those combinations that show merit in combination.

A more advanced crossover that has been shown to be much less destructive to schemata of longer definition length while being just as aggressive in recombining short defining-length schemata as other types of crossover is the two-point crossover. The two-point crossover selects two random cut sites on the string pair and exchanges the bits between the cuts. In this case, the string is arbitrarily long and can exchange any region of the strings—from a single bit at any location on the strings to the entire strings.

The final type is the multi-point crossover, which swaps random bits along the length of the string. In this way, schemata are not restricted to being composed of bits that lie adjacent to one another. This method suffers in that some additional effort is required for the operation. The others require the selection of cut sites and the movement of bit strings, while this one necessitates the generation of a random number for each bit of the string and a test of the bit in each crossover location.

These three methods have been studied extensively in prior research [7],[17]. All literature has shown the superiority of (2) and (3) over the single-point crossover. However, no significant difference has been shown between them. Therefore, a two-point crossover was chosen to be used exclusively in

this thesis due to its simplicity and speed advantage over the multi-point crossover.

Crossover rates have been shown by studies elsewhere to provide peak performance in a consistent range of values across different problems. De Jong (1975) [7] showed that crossover rates of approximately 0.60 were the most beneficial to the problems he dealt with, while Grefenstette (1986) [17] held that 0.95 was more appropriate. A very comprehensive study by Schaffer, et al. [17] led to the conclusion that crossover rates in the range 0.75 to 0.95 would all produce robust results. Consistent with the general conclusions of the ga field, a value of 0.80 is used in all simulations here without loss of generality or algorithm robustness. We have not further investigated this selection.

3.7 Mutation

Mutation occurs to keep the population from converging too rapidly and to introduce small amounts of new information that may be beneficial to locating an optimum. Since mutation rates vary widely across the ga literature, this parameter is examined in this thesis to determine its benefit in fault tolerant system design and whether any conclusions can be drawn about the relationship of mutation with population size, string length, or problem complexity.

Binary mutation occurs with probability p_m to individual bits on the strings going through the reproduction cycle. When initiated, the bit is simply "flipped" to its conjugate (0 to 1 or 1 to 0).

Having a high mutation rate keeps population diversity high. The population is not allowed to converge toward the best members because of the continual influx of new genetic diversity from the mutations. This is disadvantageous at excessively high rates because the later part of a ga simulation is composed of mixing and matching strings that have already proven to have beneficial schemata. The ga is therefore just trying to find the right combination of parameters from these good strings to make an optimal string. Continually adding new diversity into this process by an excessively high mutation rate will slow the final convergence.

At the other extreme, having a very low mutation rate allows the population to converge very fast. No additional genetic information is provided

once the initial population is established, which makes the creation of the initial population and the choice of random number generator seeds very important. Having some mutation provides gentle sanity checks on the algorithm to help prevent the algorithm from prematurely converging on a less-than-optimum solution.

3.8 Function Evaluation

The evaluation of a string's cost value is a straight forward operation. The string is simply transformed from its binary representation into the form recognizable by the cost function and the cost is calculated. This step of the ga unfortunately requires the most significant single chunk of time (computational effort) in the ga process. The time required for a single evaluation of a problem cost function is obviously constant across different optimization techniques as well. As such, the number of cost function evaluations performed is treated as a representation of time for comparing the speed and efficiency of optimization methods.

Any modification of an algorithm that reduces the number of cost function evaluations required to reach a quality solution positively impacts performance. The ga methods of this thesis only evaluate the cost function of those strings that have actually been changed by the reproduction cycle. Saving the costs of unchanged strings makes a linear increase in performance equal to the number of unchanged strings each cycle. With low mutation probabilities and a crossover rate of 0.80, about 20 percent increase in performance is achieved through the use of this feature.

4.0 Steady-State Genetic Algorithm

The steady-state genetic algorithm (ssga) is an independent development of this author created for more rapid ga problem convergence without sacrifice of solution quality. The ssga presented in this thesis advances the state-of-the-art in single criterion ga optimization without dealing with advanced ga operators or incorporating problem-specific information.

Like the tga, the ssga optimizes single criterion problems, but it differs in that the concept of a "generation" is eliminated. The ssga population continuously "evolves" to improve its average. By eliminating distinct generations, the ssga can use the improvements created by the reproduction cycle immediately, instead of waiting until a full new population is created. The best members "float to the top" and are propagated at a higher rate while the poorest members "sink" and are "killed" instead of being allowed to remain until a new generation is created. These operations allow more aggressive learning rates without unduly endangering highly fit schemata that already exist in the population.

Other forms of the ssga have since been uncovered in recent literature that incorporate much of the same fundamental logic used here. Davis [3] cites Darrell Whitley as the first to introduce a non-generational ga into the ga literature in 1988. Gilbert Syswerda coined the phrase "steady-state reproduction" in 1989 to describe the ssga evolution process, and the name "steady-state" has since become part of the ga nomenclature.

The form of the ssga developed for this thesis is shown is shown in Figure 4-1.

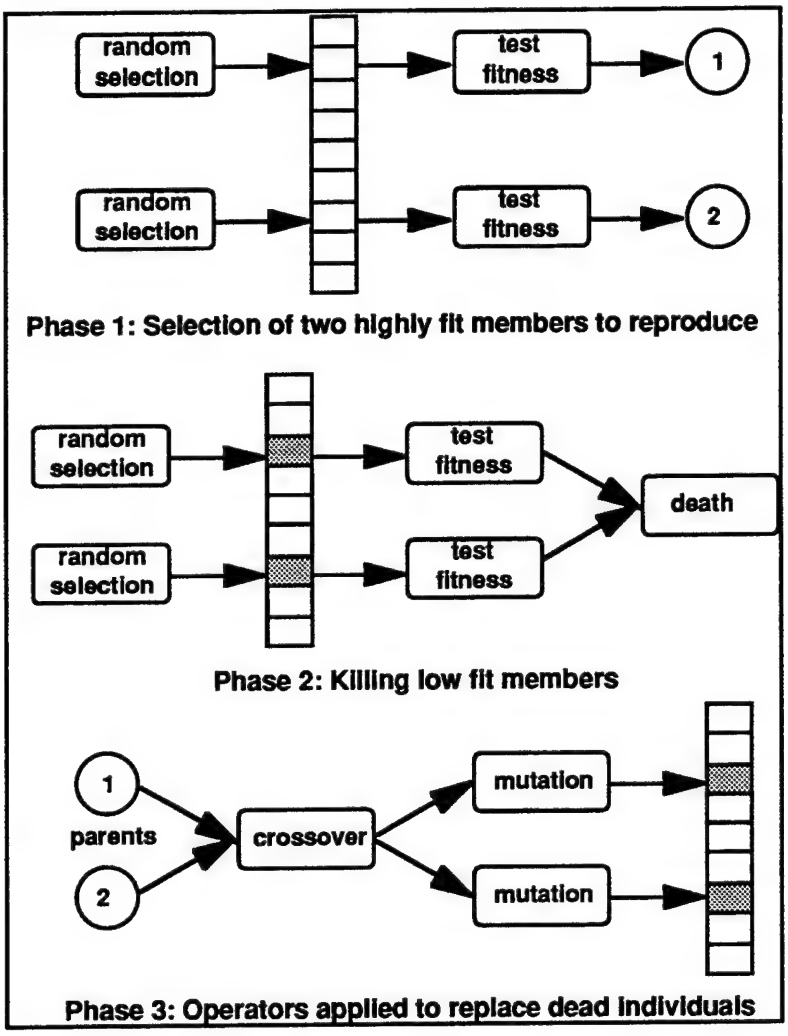


Figure 4-1: Steady-state genetic algorithm reproduction cycle

A generic ssga can reproduce any number of individuals from 1 to P in a cycle. Reproducing a full population size (P) make the ssga the same as the tga. The size of the reproduction set is also called "generation gap", and was used in the earliest research of the ga field [7]. The ssga developed for this thesis uses a reproductive set of two, to maximize reproductive turnover without hindering the crossover operation which requires two parental strings.

The ssga used in this research differs from those specified in much of the literature in that the two members that are killed are selected at random and chosen by the inverse of their fitness. This gives the poor members that have "sunk" a small, but finite chance for survival to maintain greater population

diversity. Other ssga methods simply kill the two worst members [3], thus ignoring the uncertainties involved in nature itself.

The same inverse-linear fitness scaling is used to accomplish string competition. In the tga, the fitness scale is updated each new generation when the new population is ready to begin mating. The ssga, on the other hand, updates the fitness scale anytime a newly created string has a fitness value that exceeds the bounds (highest and lowest fit members) of the current population.

The ssga of this thesis does not use "steady-state without duplicates" as is described in [3]. The inclusion of clones (duplicate members) in the population allows the ssga to gauge convergence (see Chapter 6) at some expense to computational effort, though a reasonable attempt (see below) is made at each reproduction cycle to select unique parents.

Each selection cycle (phase 1 of Figure 4-1) uses cloning as an additional selection feature. The selection of parents is based on (1) their individual fitnesses and (2) that they be unique (not duplicate strings). In the present implementation, if P (population size) prospective parents are tested for uniqueness and two unique strings are not found, then reproduction of duplicates is allowed.

5.0 Constraints

5.1 Function Constraints

The problems dealt with in this thesis are typical in most respects to those encountered when doing fault tolerant system design. However, they do not address one aspect that a designer is likely to face. Many real-world problems contain one or more functional constraints that must be satisfied. Many applications of genetic algorithms have a great difficulty with constrained problems due to a lack of a general methodology for handling constraints. Constraints are usually classified as inequality or equality. The manner in which inequality constraints are dealt with in the genetic algorithm are described first.

Inequality Constraints

Several means of dealing with inequality constraints have been attempted by previous ga work. References [11] and [3] provide details for five approaches to dealing with constrained problems in ga's:

1) throw away infeasible solutions and regenerate new strings
2) use decoders and repair algorithms
3) develop specialized data structures and genetic operators
4) use penalty functions

The first option of throwing away infeasible solutions and repeating the crossover and/or mutation operations until a feasible solution has been

generated is inefficient and generally ineffective. It guarantees feasibility at high cost to computational effort and solution quality. Ignoring infeasible strings often leads the ga to ignore constrained optima due to the indirect penalization of the region surrounding the constraint.

Decoders guarantee the generation of a feasible solution by using special representation mappings. Repair algorithms "correct" infeasible solutions, moving the solutions into the feasible region in some pre-defined manner. Both of these methods have received limited application because of the high computational effort required and the lack of generality in their implementation. Decoders and repair algorithms work reasonable well according to [11], but are highly problem specific.

Experiments cited in [11] indicate the potential usefulness of specialized data structures and genetic operators for dealing with constrained optimization problems. Problem tailored data structures combined with appropriate "genetic" operators are used to "hide" constraints presented in the problem. This approach appears to work well in the experiments performed to date, but they raise concerns due to the highly problem specific tailoring they require and the fact that many of the "genetic" operators have no biological parallel.

Finally, the application of penalties to constraint violations is an approach with wide applicability across the field of optimization. This option allows the generation of potential problem solutions without regard for constraints, and then penalizes them by decreasing their "value" in some manner. All the penalty approaches that we found rely on penalization of the cost functional.

A penalty function augments the cost functional of the problem to introduce cost degradation. It transforms a constrained problem into an unconstrained problem by penalizing constraint violations [7]. For example, the original constrained problem has the form:

$$\begin{aligned} &\text{Optimize:} && J(\mathbf{x}) \\ &\text{Subject to:} && g_i(\mathbf{x}) \le 0; \qquad i = 1, 2, \ldots, m \end{aligned} \tag{1}$$

This can be transformed to the unconstrained form:

$$\begin{aligned} &\text{Optimize:} && J(\mathbf{x}) + \Phi[g_i(\mathbf{x})]; \qquad i = 1, 2, \ldots m \\ &\text{where} && \Phi \text{ is the penalty function} \end{aligned} \tag{2}$$

The penalty can be additive, multiplicative, or otherwise, and can either apply direct penalties to all violations or use some scaling to discriminate between degrees of constraint violation (usually increasing the penalties for larger distances from feasibility). Davis [3] cites evidence that genetic algorithm penalty functions, especially those that discriminate between different types of infeasible solutions, are competitive in performance to methods that use specialized operators without loss of generality. The major drawback of penalty functions, however, is the necessity to develop a separate appropriate scaling for the cost functional of every problem attempted.

General guidelines for the construction of ga penalty functions are listed in [15]. The work concludes that for ga problems having few constraints and a limited set of full solutions (such as the discrete problems of this thesis), penalties that are solely functions of the number of violated constraints are not likely to find solutions. Good penalties, according to that work, are functions of the distance from feasibility for each constraint violation. Reference [15] asserts that a ga should incorporate what it calls the *completion cost* to generate the best penalties.

Completion cost refers to how much of a cost difference would have to be accepted to make an infeasible solution feasible. Instead of being rejected or labeled "undesirable", infeasible solutions should be thought of as *incomplete* solutions. The completion cost helps determine the appropriate penalty to assign based on the "incompleteness" of its structure.

The completion cost used in [15], though it has a good deal of theoretical appeal, requires *problem-specific* estimation of the tradeoffs between constraints and the cost functional. The difficulty and tedious nature of this estimation has led the author to create a simple method of applying function constraint penalties that capitalizes on the function knowledge contained inherently within the genetic algorithm population. A multiplicative penalty function that degrades string fitness (F) is used to shown the applicability of this new approach.

Automatic penalty functions

To justify the use of a penalty that does not require problem specific scaling, we look at a general single criterion problem with a single inequality constraint. This problem is the minimization of a cost function (J) subject to the constraint $g < g^*$. The population of the ga appears similar to that of Figure 5-1 if the constraint, shown at $g = g^*$, is ignored.

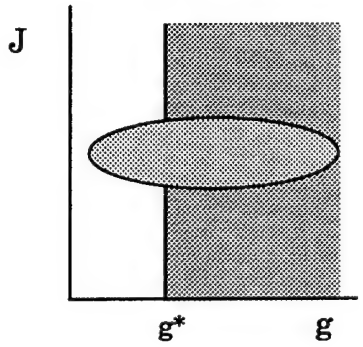


Figure 5-1: General ga population distribution with constraint bound shown

The values of J would have started in some unknown range larger than that shown, but the evolution of the population toward an optimum has narrowed the ga's focus to the given J values. The values of g, on the other hand, have not been directly considered, and a wide range of their values for the current range of J exists in the population.

As mentioned earlier, references [3] and [15] state that penalty functions should account for the distance from the constraint bound g^*. Unfortunately, these penalty functions have to be applied *problem-specifically*. The DM is forced to decide the appropriate scaling of g.

Let's explore the issue of *problem-specific* penalty functions a bit further by way of a comparison of possible g_1 values, where $g_1 > g^*$. Assume that the distance from g^* to g_1 is $\delta g = g_1 - g^* = 10$. In one theoretical problem, the actual constraint is a distance tolerance of $g < 2$ cm, where $\delta g = 10$ is an intolerable violation. In another problem with $\delta g = 10$, however, the constraint is $g <$ 10,000 cm and $g = 10{,}010$ is actually very close to the optimal feasible solution in the design space. The problem is that optimization algorithms do not known the difference between 10 cm for one problem and 10 cm for another. As such, the penalty on g must be defined *problem-specifically*.

However, another approach may exist. It is the assertion of this author that the genetic algorithm contains sufficient information about the "constraint space" of any problem to assign its own approximation of "closeness"!

The ga works from a whole population of points. Going back to Figure 5-1, the ga knows the g values of the strings of its present population. With a properly sized population, sufficient diversity of values should exist in the current population to allow the ga to make effective estimates of "closeness" to the constraint bound (g*). The premise of this assertion is that the DM himself frequently estimates "closeness" in forming the penalty scaling for a problem and usually uses knowledge of the *probable* g values in that estimate (i.e. for g < 2 cm, $\delta g = 10$ is bad, but for g < 10,000 cm, $\delta g = 10$ is acceptable).

To incorporate the ga's knowledge of the constraint values into an actual penalty formulation, an assumption must be made of what *relative* distances would generally be considered appropriate by a DM. The assumption made for this thesis is that any distance into the infeasible region (δg) less than the range of feasible g values in the *current* population is acceptably "close" to warrant involvement in the reproduction process. To form a penalty from this assumption, the distance (Δg) is defined as:

$$\Delta g = g^* - g_{min} \tag{3}$$

where g_{min} is the minimum g value of the strings in the current population. The remainder of this chapter details the particular penalty function used in this thesis to apply this general ga "hands-off" penalty approach.

Since a penalty that accounts for the distance from the constraint bound is preferable, the amount of penalization to be allowable at $g = g^* + \Delta g$ must be defined. This thesis assumes arbitrarily that the penalty should increase from 0% at $g = g^*$ to approximately 90% at $g = g^* + \Delta g$. The exact form of the penalty function is arbitrary, as long as it exhibits the desired trend, i.e. it decays as a function of the distance (δg) that a value is from g*.

We apply a decaying cubic of the distance (δg) relative to Δg in this thesis:

$$G(g(\mathbf{x})) = \begin{cases} \left(\dfrac{g^* - g(\mathbf{x})}{\Delta g} + 1.0 \right)^{-3} & \text{if } g(\mathbf{x}) \geq g^* \\ 1.0 & \text{otherwise} \end{cases} \tag{4}$$

where $G(g(\mathbf{x}) = |\Delta g|) = 0.125$ (1 - 0.125 = 87.5% penalty). The form of the fitness penalty G is shown in Figure 5-2.

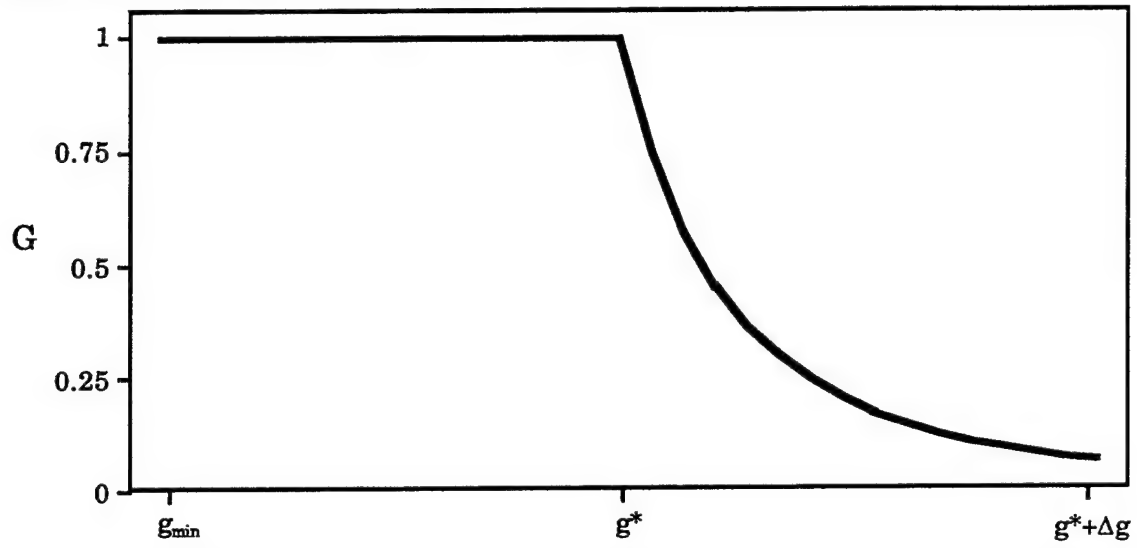


Figure 5-2: Penalty function (G) as a function of the amount of violation

G can now be applied as a multiplicative *fitness penalty function* that degrades fitness (F) as a function of the amount of constraint violation:

$$F' = GF \tag{5}$$

Figure 5-1 assumes that g and J are very weakly correlated, such that a change in g has little or no effect on J. In many real-world problems, however, the constraint and cost function are negatively correlated as in Figure 5-3. Though the range of J narrowed as before, the shape of the population points in the g-J space shows the conflicting relationship between g and J.

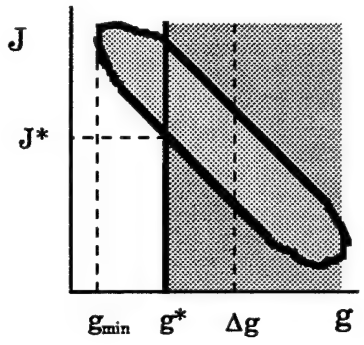


Figure 5-3: Population distribution for correlated cost and constraint

This figure has the appearance of a two-dimensional multicriteria problem. The similarities will be addressed and taken advantage of in Chapter 15 with the ε-constraint method of multicriteria optimization.

For the example in Figure 5-3, the optimal solution lies at [J*, g*]. If no infeasible points are allowed in the population, no *incomplete solutions* containing the schemata of J* are allowed, while schemata of J > J* have little constraint on their allowed frequency in the population. This effect is shown by plotting constant values of J (called isoquants) versus g:

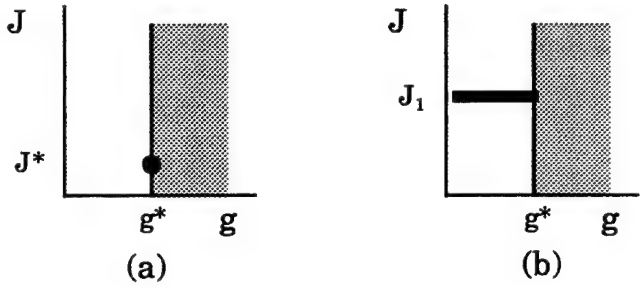


Figure 5-4: Constant J value isoquants when infeasible points are ignored

Figure 5-4a shows that the only solution possible with all the schemata of J* is the optimal solution. Its chances of appearance in the population are greatly reduced by this strict approach. On the other hand, an isoquant of a J_1 > J* in Figure 5-4b allows a great many variations of J_1 schemata to appear in the population.

Applying G to the problem makes the isoquants above take the form of Figure 5-5. The isoquants of normal J values are plotted in the J′ space , where J′ is the cost of strings that have G applied to their fitness (F).

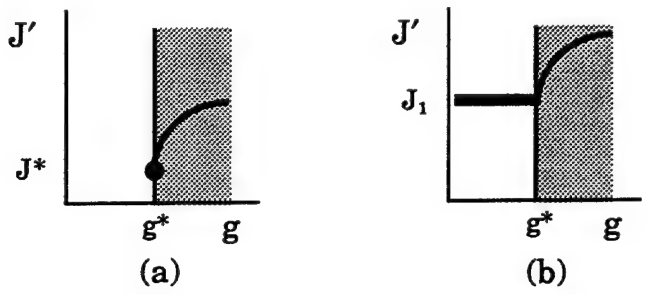


Figure 5-5: Constant J isoquants with G penalized infeasible points

Though penalized, J* incomplete solutions are allowed into the population so that the ga has a much better probability of generating competitive schemata of J* and thus locating [J*,g*]. G allows fitnesses of J* to be competitive with non-optimal J with $g < g^*$ so that the schemata of [J*,g*] have a higher probability of occurrence.

For the negatively correlated g-J example, let's look at the effect G on fitness (F) for different g*. Figure 5-3 shows g* close to g_{min}. Applying G to this example and looking at F as a function of J for possible isoquants of constant g value gives the form:

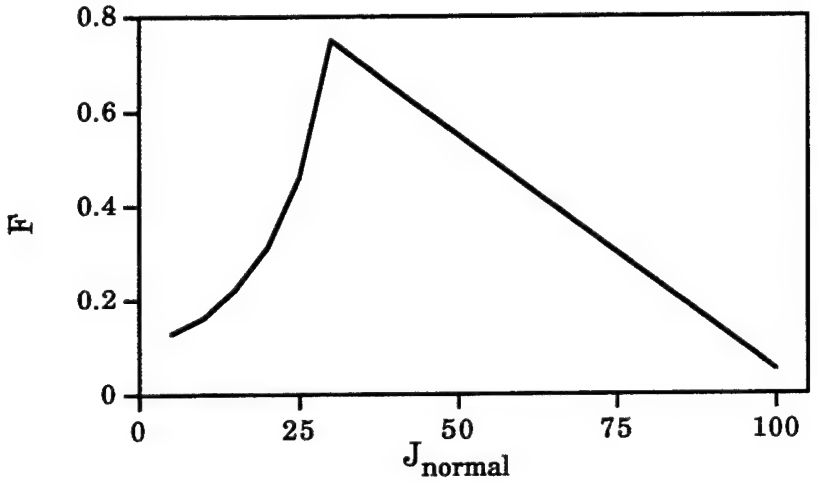


Figure 5-6: Penalized fitness function for g* close to g_{min}

F is degraded for constrained points. Because g* is located close to g_{min}, all J values are allowed to reproduce (assigned $F > 0$), but high penalties are assigned for very large distances (δg) from g*, *even though the change in J is very small.*

Next let's look at the same example when g* is placed close to g_{max}.

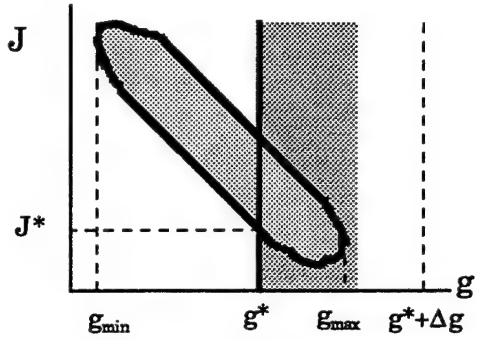


Figure 5-7: Population distribution for correlated J-g with g* far from g_{min}

The fitness with G applied takes the form in Figure 5-8:

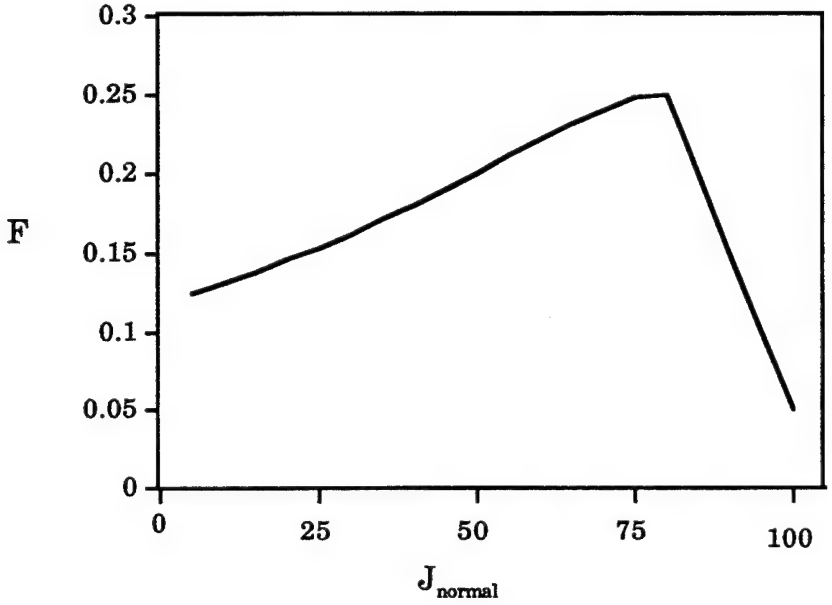


Figure 5-8: Penalized fitness function for g* far from g_{min}

The span between g_{min} and g* is large in this example. As such, G does not heavily penalize g > g* values it finds in its population because it assumes that these values of g are sufficiently "close" to g* to warrant their inclusion in the population for the potentially good schemata they may contain.

The key observation from Figure 5-6 and Figure 5-8 is that since fitness provides a ranking of strings relative to one another for parent selection, the penalty not only keeps the fitness of infeasible points degraded even if they have good cost function values but also allows infeasible points to be

competitive to the feasible region based on the *incomplete cost* of their schemata, which correlates to the peak performing penalty functions suggested in [15].

Elitism must still be applied in conjunction with this fitness penalty to ensure that the solution is always feasible. Again, we note that the actual choice of applying a multiplicative penalty to fitness is just one way of approaching inequality function constraints in genetic algorithms. What is novel here, however, is the use of the ga's knowledge of the "constraint space" to scale the penalty automatically.

Equality constraints

Equality constraints have not been dealt with directly by this research, but the ideas presented for inequality constraints can be used for many equality constraints the user may deal with by simply penalizing deviation from the function equality.

In most instances, however, the existence of equality constraints strengthens the designer's ability to solve the problem. The existence of an equality function constraint is most often used to "reduce the order" of the problem. The equality constraint is usually presented as some function of the design parameters, which allows the user to solve for one variable as a function of the others. This solution is then placed into the remaining functions of the problem, thus reducing the number of free design variables that must be solved for. The order reduction calculation may be a simple algebraic equation in the case of linear problems, or it may have to be solved numerically if the problem is nonlinear (as in this context). In either case, the solution to the intermediate step of order reduction almost always reduces the computational requirement of the cost functionals.

5.2 Parameter Constraints

Apart from constraints placed on the cost functionals, a far more typical instance involves constraints of the parameters themselves. However, the ga deals with this issue directly. This is a consequence of the ga's use of parameter coding rather than the actual parameters.

The ga user can apply such constraints directly as he or she inputs the parameter values into the problem. An equality constraint is simply a fixed value of the parameter. Inequality constraints are treated as hard bounds by limiting the range of the parameter to the feasible region. This is done by scaling the binary word which represents the parameter in the string to parameter values which just span the allowable range. The ga operates directly and solely on the domain provided. By not using derivative or other auxiliary information, the ga never exceeds parameter bounds.

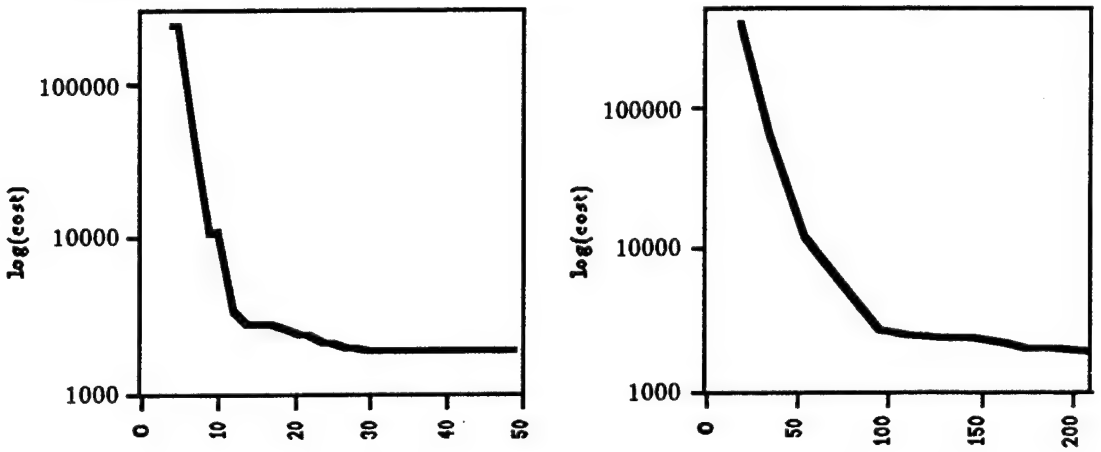
6.0 Convergence

When one talks about convergence, the issue is twofold. First, convergence refers to how near the solution obtained is to the actual minimum of the problem. Secondly, one is interested in the method's approach to a solution—how rapidly in time a solution is obtained. Therefore convergence constitutes two criteria by which the ga can be compared to other optimization methods.

6.1 The Fundamentals of Convergence

The speed at which a solution is reached is important, but in the ga it demands a tradeoff. Getting the solution as soon as possible is obviously beneficial, but if the population converges too quickly, diversity is lost before the ga has had a chance to make a reasonable search of the design space. Fitness testing, elitism, and a properly sized population maintain this trade-off for peak performance. As described in Chapter 3, fitness scaling is structured such that poor members may still inject genetic material into the mating pool, though at a lower rate than highly fit members. Elitism describes the process of always keeping the best member ever created in the current population to ensure that its schemata are kept available. This mechanism prevents crossover and mutation from destroying the best genetic code found so far. The importance of the correct population size is examined in Chapter 8. However, it is safe to note here that using a population size too small limits the diversity available, while using a population size too large excessively slows convergence without necessarily increasing in solution quality.

The single criterion ga has the beneficial quality that it *always* has a population size (P) number of candidate solutions on hand. Therefore, a solution is always available to the decision maker (DM) at any stage of operation that termination is desired. To illustrate more clearly how the ga converges on a problem, Figure 6-1 shows how the Down-hill simplex and steady-state genetic algorithm (ssga) operate on the Warning Lamp problem.



(a) lowest cost in simplex (b) lowest cost in ssga population

Figure 6-1: Convergence to a Warning Lamp solution

The most obvious observation from this figure is that the shape of the two plots is very similar—this is perhaps the most encouraging observation as well. The Down-hill Simplex method is directly suitable for the Warning Lamp problem . This solution space is completely monotonic with a steep gradient over most of the surface, and a benign gradient near the minimum. Since we wish to prove in this thesis that the ga is a very robust method, to ask the ga to outperform the Down-hill Simplex on this problem is unrealistic. The Down-hill Simplex is able to reach the minimum of this problem (which implies zero error in its ability to terminate in the proximity near the global minimum) in under 50 cost function evaluations (a bull's eye for efficiency!) no matter what starting point is used.

The ssga in Figure 6-1b is able to reach a solution within 15% of the true minimum in about 200 cost function evaluations. The 15% value represents continuous solutions near the optimum. In doing continuous optimization with a ga it is necessary to verify the discretization of the continuous parameters is sufficiently fine to allow the ga to find the optimum. In this example, the

resolution is adequate to allow better solutions to be found, so that the poorer performance indicates the difficulty the ga has in assuring convergence to the optimum of continuous problems. This result shows that the ssga is competitive even when operating outside its preferred conditions. This analysis was done with a population size of 20 to show the convergence better—thus the performance could be improved by using the optimal parameter settings described in Chapter 8. Also, this problem was purely continuous, and the ga operates best when the design space is limited in an integer or discrete manner.

This genetic algorithm uses elitism, which means that the best member ever generated in a run is always kept in the current population. The ga converges the whole population toward the elite members, but genetic algorithms provide no guarantees of convergence on arbitrary problems. To say that the ga is only capable of ensuring a solution within 15% of the minimum would disturb some decision makers (DM). The ga does sort rather quickly through a great deal of the space looking for interesting segments, but it is nonetheless a very "coarse" method, without any real guarantees. However, this must not be seen as a limit to the genetic algorithm's capability. More rapidly convergent methods, such as the Down-hill Simplex method, sacrifice global minimum assurance and problem flexibility even when coupled with simulated annealing. As a result, ga's can be used in situations other methods shy from or are helpless to proceed in. If a guarantee of convergence is desired, the DM should use the ga to find the local region of the optimum and then use another method to fine-tune the solution to her specifications. This approach allows the DM to combine the global search ability and flexibility of the ga with the convergence behavior of a local technique.

6.2 Steady-State Genetic Algorithm Behavior

The bottom line of convergence rests with the quality of the solutions obtained. This will be analyzed in great detail in the next two chapters, but we will first take an introductory look at the genetic algorithm convergence on a difficult problem. First, let's look at the convergent behavior of the ssga. Figure 6-2 shows the mean (average) cost of the entire population as a function of cost function evaluations (cfe) when the ssga is applied to the TISS problem. The first plot shows the ssga with a population size (P) of 60, while 200 is used in the second.

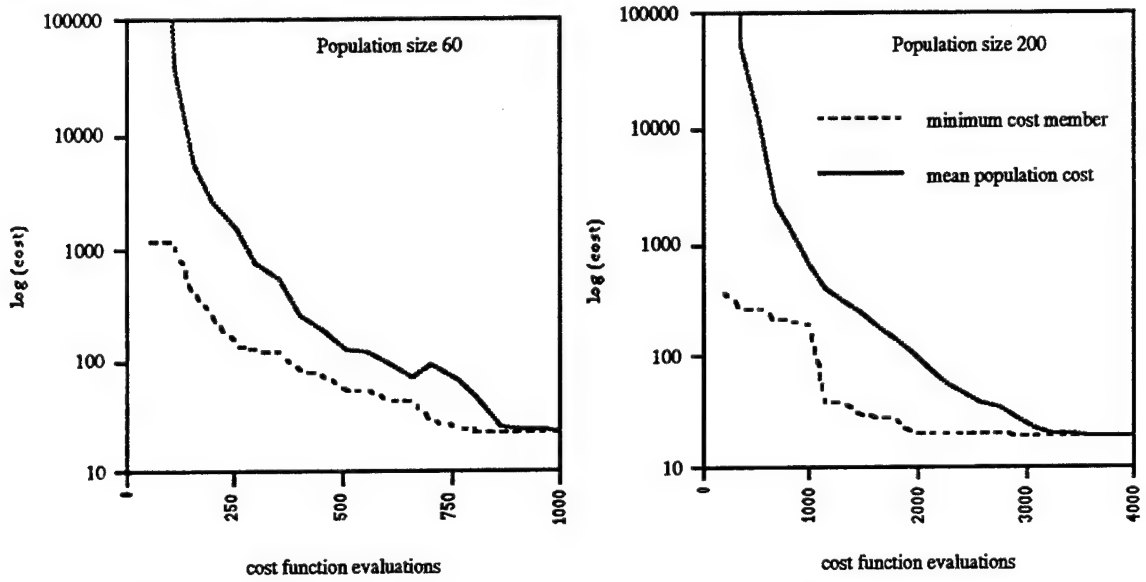


Figure 6-2: ssga convergence behavior on the TISS problem

Both plots show excellent convergence characteristics. The convergence of the first occurs in about 1,000 cost function evaluations and the second in less than 4,000 cost function evaluations. The average and minimum cost function values lie at about 30. Note that the general performance of the ssga, as shown by the plot shape and solution attained, does not change a great deal even when the population size is changed dramatically—the effect is only seen in the cfe required for population convergence. This illustrates the robust nature of this technique.

6.3 Traditional Genetic Algorithm Behavior

Unlike the ssga, the tga does not show the same insensitivity to population size. Figure 6-3 shows the different convergence behavior of the tga population when P is changed.

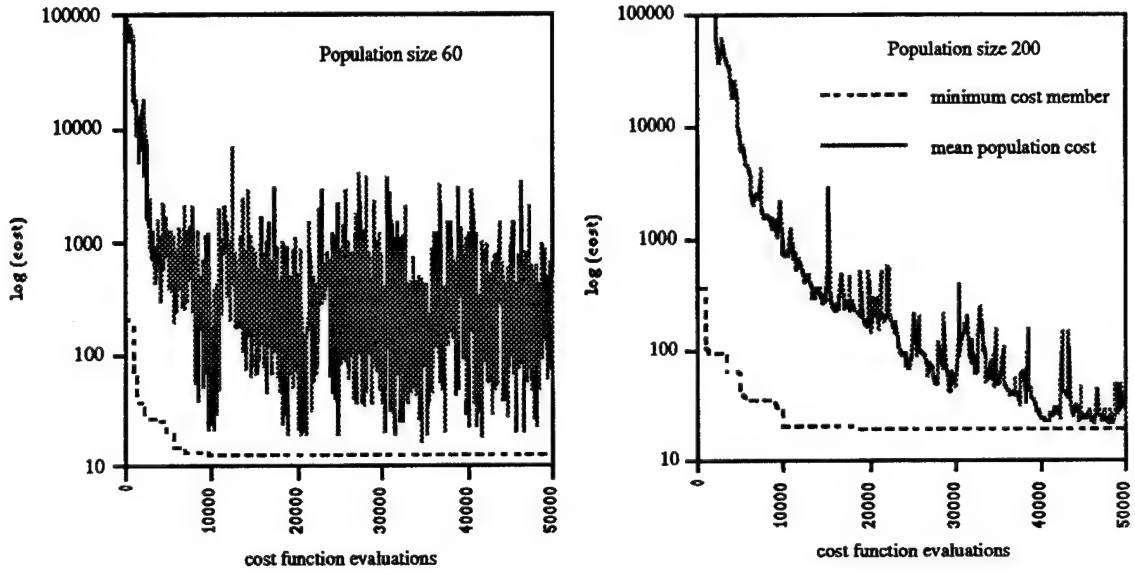


Figure 6-3: tga convergence on the TISS problem

The first plot shows the behavior of the tga when P is near optimal as described in Chapter 8. Even at the optimal P, though, the mean cost does not make the smooth transition seen in the ssga. The fluctuations are extremely erratic, and though the minimum cost member does not change after 10,000 cfe, the population fluctuates widely until termination at 50,000 cfe.

The second plot, with a tga population size of 200, illustrates a different tga convergence behavior emerges as P changes. The general form of this plot is much like that of the ssga. It differs however, first in that the lowest cost member is slightly higher (worse) than the first plot of Figure 6-3 and both plots of Figure 6-2, but even more prominently in that it requires almost 50,000 cfe for the mean cost to converge to the lowest cost member.

Going back to the two criteria for good convergence, the tga achieves its minimum value in more than twice as many cfe as the ssga. The average cost of the small tga population size doesn't show any convergence behavior compared to the ssga and large population size tga, but it gives a solution of better quality than the larger population size tga. The final test of convergence

of these two methods will come when the solutions they generate are compared to those generated by the Branch and Bound method in Chapter 8, but it can be seen here that, in general, the convergence of the genetic algorithm exhibits quantitatively favorable characteristics even on the difficult TISS problem representative of fault tolerant design problems of moderate to high complexity. The convergence behavior of the ssga is especially appealing in that the constantly "evolving" population does not experience the erratic behavior experienced by the tga with its separate generational populations.

6.4 Termination Criteria Analysis

What the above discussion of convergence means in actual ga implementation, however, is that it is very difficult to know when to terminate the ga search. In practice, the user must usually limit the computational effort because of monetary or time constraints. As such, the ga is ready to provide a solution any time the user needs one, but the user still would like to know the relationship between computational effort and solution reliability.

Population convergence does indeed occur for intelligent implementations of the genetic algorithm. At some point in an algorithm run, diversity falls below the "critical mass" necessary to fully exploit the design space. When this occurs, if the user still wishes optimization to continue, diversity should be re-injected into the population. As part of this thesis, we attempted to identify mechanisms that could illustrate ga convergence and reasonably indicate that termination should occur.

To accomplish this goal, convergence/termination testing was done on the basis of 8 candidate ideas of convergence developed for this research:

1) the number of cfe between changes of the lowest cost member
2) the difference between average and lowest population costs
3) an indication of population variability calculated as the range between the highest and lowest cost members divided by the mean population cost
4) the variance of the population costs
5) the average variance of the parameter values in the string
6) the product of the parameter variances
7) the average of the "bit likeness"
8) the product of the "bit likeness" (PBL)

Moderate testing was accomplished using all of the test problems to determine which of the eight provide the user some quantifiable sense of convergence. Before the results of this analysis is shown, a quick discussion will reveal and explain those ideas that were quickly eliminated.

Options 4 and 6 were immediately discarded because they were discovered to not provide an adequate desirable stopping point indication. Though the characteristics of these figures of merit did change over the course of a ga run, there did not appear to be a strong correlation between a desirable convergence indication and their characteristics.

The number of cost function evaluations (cfe) between changes in the lowest cost member (Option 1) seemed to have a great deal of merit when the plots of this item were first obtained. However, it was determined that it is impractical for two primary reasons. First, greater numbers of continuous parameters provide for a larger design space that can often cause changes in the lowest member that are insignificant to amount of effort required. This also occurred in discrete problems with shallow minima. The second reason dealt with problem complexity. In the TISS problem, for instance, 10,000 cost functions could easily be required, and the number of cfe between changes of the lowest cost member could best be set at 500+, while the TRIPLEX problem would only need a cfe change of about 50 for the small number of total cfe required. These issues make this idea potentially workable, but possibly too complex to implement for the common user.

Most of the other ideas prove inadequate due to the amount of noise in the measurement or to scaling difficulties. Noise in the measurement is especially evident in Options 2 and 3. Option 3 is just a scaled version of 2, and attempts to approximate convergence by the cost relationship: (high-low)/mean. Figure 6-4 shows that because the population possesses a great deal of internal diversity even when the average and lowest cost values are nearly the same, the figure of merit does not show measurable convergence.

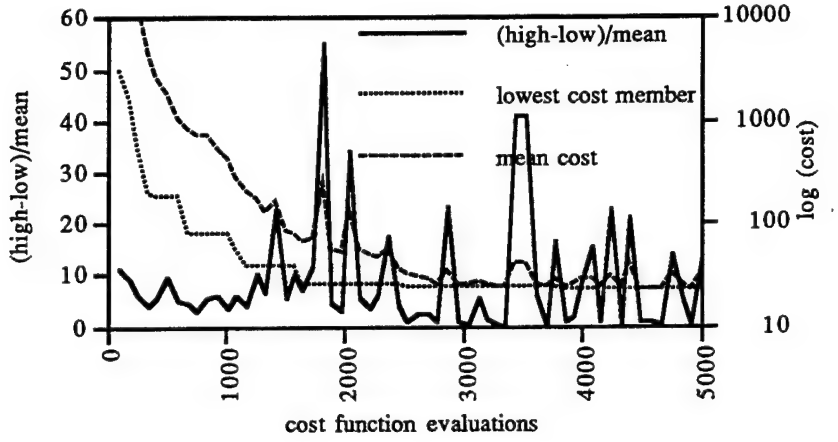


Figure 6-4: Convergence method candidate: (high-low)/mean

This figure shows an ssga run for the TISS problem solved using a 100 member population with crossover and mutation rates of 0.80 and 0.02, respectively. The simulation shown is run for 5,000 cost function evaluations (cfe). The figure shows three lines: the mean population cost, the lowest cost (best) member of the population, and the figure of merit. The two costs are associated with the right-side axis, while the left side is scaled separately for the figure of merit. This run is typical for ssga (and most tga) performance in that the lowest cost and mean cost lines decay and, though initially separated, the lowest cost reaches steady-state and the mean cost decays to meet it. Somewhere between 2,000 and 4,000 cfe some means of determining adequate population convergence must exist, but Figure 6-4 is not such a measure.

Another idea that initially seemed to show merit was Option 5, the average of the parameter variances, which has the form:

$$f_5(\mathbf{x}) = \frac{1}{n}\sum_{j=1}^{n}\left(s_j^2\right)$$
$$\text{where} \quad s_j^2 = \frac{1}{P-1}\sum_{i=1}^{P}\left(\bar{x}_j - x_{ji}\right)^2 \tag{1}$$

where n is the number of design parameters (x) of the problem and P is the population size.

The behavior of this figure of merit is shown in Figure 6-5 for the same TISS problem run used in Figure 6-4. Using the mean parameter variance as a guide, we can see that full population convergence occurs at 3,000 cfe.

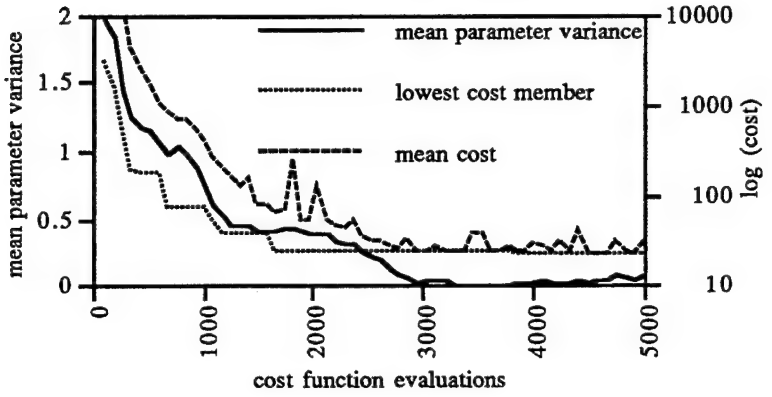


Figure 6-5: Convergence method candidate: mean parameter variance

This option appears to give a good indication of convergence, but its problem is one of scaling. The left-side axis shows that the mean parameter variance decays from about 2.0 to zero in this instance. This range, unfortunately, cannot be generalized beyond this example. As a consequence, if the user wishes to terminate the ga prior to *complete* population convergence, this option does not provide a simple quantifiable convergence limit.

Options 7 (mean bit likeness) and 8 (product bit likeness) are a return to the fundamental ideas of the ga. The ga operates on genetic binary strings of data that are independent, but uniquely correlated to the problem of interest. Using the assumption that a population with qualities X is converged no matter if problem A or problem B is being optimized, it is only natural that the quest for a convergence criterion should look solely at the bit strings.

The term "bit likeness" is used to refer to the fraction of the bits that are the same at a bit location (locus) across the whole population. For instance, look at the following population of 4 strings of length 3:

string 1	1	1	0
string 2	0	0	0
string 3	0	1	0
string 4	1	1	0
bit likeness	0.50	0.75	1.00

Table 6-1: Bit likeness illustration on 4 member population

From this illustration the meaning of bit likeness is easy to see. The

first locus has a bit likeness of 0.5 because there are equal numbers of 0's and 1's. The second locus has a value of 0.75 because 3/4 of the bits are 1's, while the last locus has all 0's and thus has a bit likeness of 1.00. The bit likeness is the greater of either the fraction of 0's or 1's at each locus. This convention is based on the assumption that the ga propagates beneficial schemata; therefore the greater of 1's or 0's at a locus should signify participation in more beneficial schemata and better population fitness.

In the population of Table 6-1, the average of 0.5, 0.75, and 1.0 gives a mean bit likeness of 0.75. The mean bit likeness is shown below for the example TISS problem.

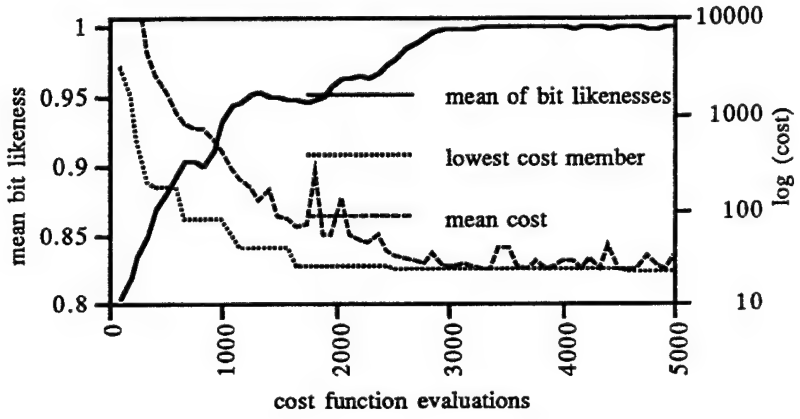


Figure 6-6: Convergence method candidate: mean of bit likeness

The convergence of this figure of merit is complete by 3,000 cfe. To reach that point (1.0), all strings must be identical. Again, absolute convergence is farther than we normally want the ga to operate to reduce redundant and unproductive operation, but as with the "variance of the parameter values" (Option 5) extrapolating a quantifiable measure of general population convergence is difficult. This figure of merit begins at values between 0.7 and 0.85 for various problems. Using the behavior of the mean cost and lowest cost lines as a guide, terminating at a particular mean bit likeness value such as 0.95 gives different and sometimes unsatisfactory results for different problems.

The final option is the product of the bit likeness. In Table 6-1, the product of the bit likeness (PBL) is (0.5*0.75*1.0 = 0.375). The behavior of this figure of merit is quite different and reveals some interesting characteristics of

population convergence useful for formulating a termination criterion. The range of PBL is always 0 to 1.

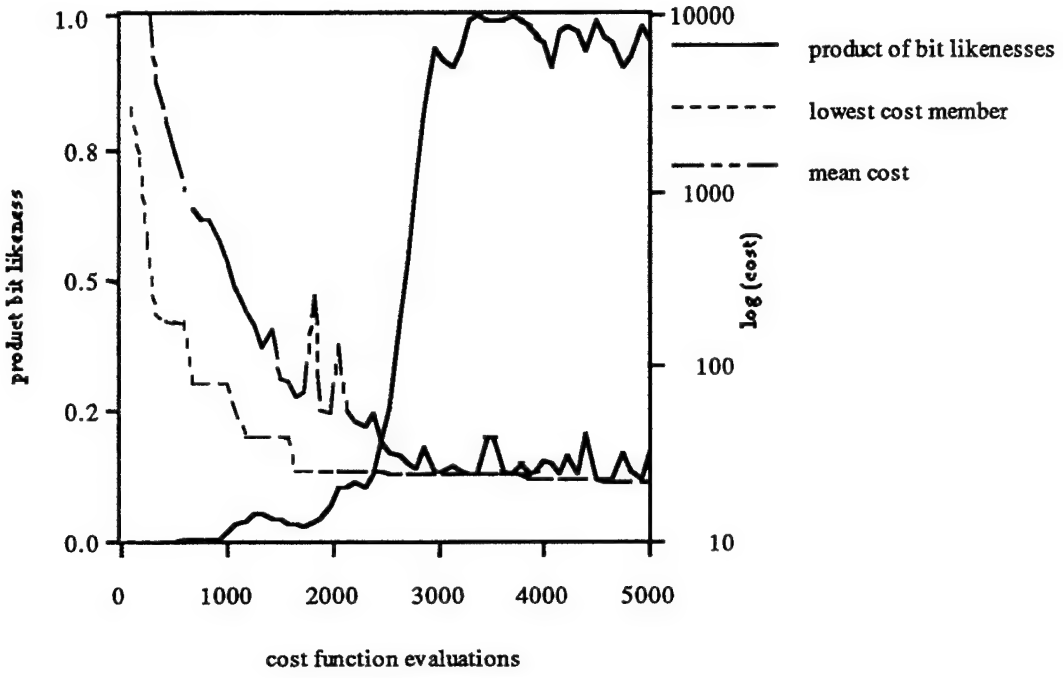


Figure 6-7: Convergence method candidate: product of bit likeness

The behavior of the PBL shown in Figure 6-7 for the TISS example appears to be typical for all the problems solved by the ssga in this thesis. PBL applies only to the ssga because of the great fluctuations in tga generation statistics. The initial values of PBL are near zero, while the values after total population convergence are near one. These ranges are understood as follows: in the initial stages, the mean bit likeness is somewhere in the range of 0.8. PBL is approximately 0.8^l, where l is the bit string length. For typical fault tolerant problems, where the string lengths are in excess of 50 bits, PBL is a very small value (0.8^{50} = 1.4e-5). Even small problems of only 16 bits have initial PBL values close to zero (0.8^{16} = 0.028).

To approach a PBL of 1.0, the population must lose a considerable amount of diversity. As the strings of the population begin to sort through schemata and choose those of higher fit regions of the design space, the bit likeness at each locus of the string is assumed to gravitate toward higher fit (better) binary values. This effect is best described by two examples.

First assume a locus on a string where the initial population has a bit

likeness of 0.7 of ones and *ones* represent high fitness and optimal solution(s). As the ga operates, the number of ones will increase, moving the bit likeness from 0.7 to 1.0 and increasing the overall value of PBL. Secondly, assume the same locus with the bit likeness 0.7 of ones, but now with *zeros* representing high fitness and optimal solution(s). As the ga operates, the number of zeros in the locus will increase, and the bit likeness will first decrease from 0.7 to 0.5 (lowering PBL) before rising from 0.5 to 1.0 (raising PBL) as the zeros become more frequent than ones.

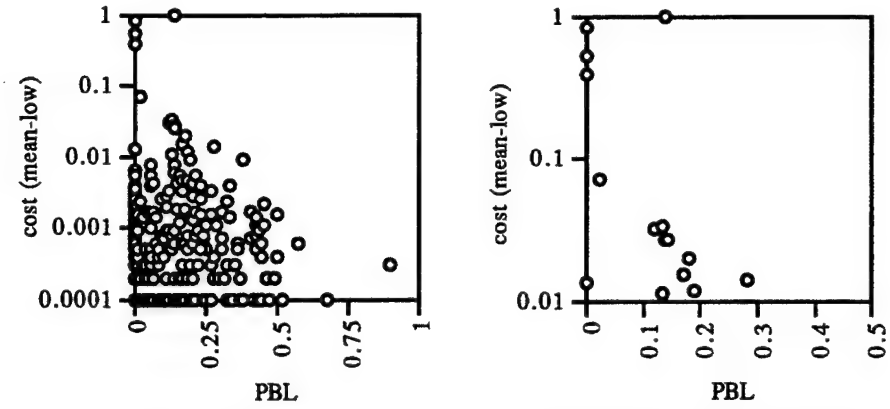
The result of many loci along a string with bit likeness behaving like either of the above two examples is that the decrease in some loci bit likeness will offset the increase in others, keeping PBL near zero for the first stages of ga optimization. The sharp rise of the PBL value occurs when all of the loci have positive bit likeness changes.

The object of this analysis is to make PBL a sufficiently general convergence criterion applicable to any string length.

There is an intuitive relationship between the length of the bit strings and the appropriate termination value for PBL. Unfortunately, since only three string lengths of 12, 16, and 51 are used for the TRIPLEX, ASYMMETRIC LAMP, and TISS problems respectively, this thesis cannot provide adequate data to determine a reliable relationship. We can, however, attempt to determine some PBL guidelines for problems similar to those of this thesis.

Figure 6-8 and Figure 6-9 show the relationship of PBL to two potentially useful quantities. The data shown is a compilation from the 180 TISS simulations performed in the population analysis section of this thesis run to 50,000 cfe (see Chapter 8). Figure 6-8 shows the difference between the mean and lowest costs as a function of PBL. The y-axis is a normalization between zero and one of all simulation result values. Simulation points are included in Figure 6-8a for normalized ranges above 1e-4.

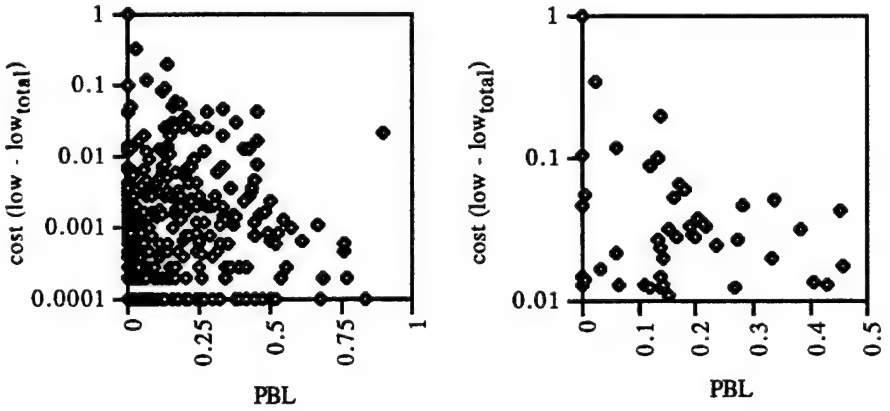
Figure 6-8b limits the axes to include only the normalized range above 0.01. This second plot shows that if the decision maker (DM) considers 1 percent of the original range between mean cost and the best population member sufficient convergence, the PBL value for the TISS problem can safely be placed at 0.3 to terminate all 180 TISS simulations performed in Chapter 8. Raising the bound to 5 percent, a PBL termination value of 0.2 can be used.



(a) full TISS analysis (b) restricted view of analysis

Figure 6-8: PBL comparison to average cost decay on TISS problem

However, as stated earlier, there is a second quantity of potential interest for determining a termination criterion. The second quantity is the difference between the present lowest cost and the lowest cost to be determined by the simulation. This quantity measures the effort involved in achieving a better solution. Unlike the previous figure of merit, this one cannot be computed during the simulation. Figure 6-9 shows a compilation of this measure versus PBL for values above 1e-4. As above, the quantities are normalized between zero and one to represent the full range of values of the TISS problem analysis in Chapter 8.



(a) full TISS analysis (b) restricted view of analysis

Figure 6-9: PBL comparison to lowest cost decay on TISS problem

Figure 6-9b is a limitation of (a) based on certain assumptions. The second plot assumes that the DM would be interested in convergence to 1

percent of the original value of this quantity. One point, seen by itself at the far right of the first plot has been ignored in the second plot. Using the information of Figure 6-9b, the DM could reasonably set a PBL limitation of 0.5 to have a great deal of assurance of achieving the best solution possible in any TISS simulation.

The introduction of PBL has given a figure of merit with a strong correlation to ga population convergence. Unfortunately, we have not directly used the PBL's particular behavior (i.e. the sharp rise from hovering near zero to hovering near one) to signify convergence. Additional work in this area that would determine a suitable test for the rise would enhance the benefits of PBL usage.

The remaining analyses performed for this thesis terminate the genetic algorithms by setting PBL levels between 0.1 and 0.3 and dictating a maximum number of cost function evaluations to perform. The mutation analysis does not use PBL to observe the effects mutation has on degenerated populations with little diversity remaining. The population size analysis does use PBL under the assumption that extra effort to ensure the best possible solution is unnecessary.

As was stated earlier in this section, PBL has only been developed as a termination criterion for the ssga. While the tga can be terminated on PBL, the termination PBL values are generally much lower because of the large fluctuations of the tga population characteristics from generation to generation. As such, PBL in the tga does not illustrate the same characteristics seen in Figure 6-7 and consequently is not recommended as a termination criterion for the tga. The fact that PBL applies only to the ssga leaves the tga without an adequate method of determining convergence.

The ga simulations performed in this thesis keep a single degenerative population because in most fault tolerant system design applications, population rejuvenation is not necessary to attain a solution with adequate quality to satisfy the DM. In limited instances not encountered by the efforts of this research, the design space may be particularly difficult to optimize or the DM may not be satisfied with the solution obtained by a single ssga optimization run. In either case, population rejuvenation may be necessary to meet the DM demands for the design. Rejuvenation occurs by using the solution attained by a completed ga simulation as one of the points of the population of a new ga simulation.

7.0 Mutation Rate Analysis

This section analyzes the effect of mutation on ga optimization of fault tolerant systems. Tests are performed to determine the optimum mutation rate for these problems. Nine probabilities of mutation are tested in the range from 0 to 1. Both the tga and ssga are applied to the TRIPLEX and ASYMMETRIC LAMP problems (see their descriptions in Chapter 2). All other ga parameters are held fixed for this analysis. Five seeds (runs) are performed for each mutation rate to reduce the result's sensitivity to the seed. All runs go to 300 cost function evaluations (cfe) to maintain the efficiency pressure established by the Branch and Bound method for these two problems. A population size of 100 is used for the tga, while a value of 40 is used for the ssga.

Two criteria are examined. First, the average population cost criterion looks at the convergence of the population to the design space optimum and indicates how reliable a result would be if selected prior to steady-state operation. This situation is directly applicable to large problem applications where steady-state may not be attainable within the time constraints allowed. The other criterion is the lowest cost during the run. This criterion measures the ability of the algorithm to find the best solution possible.

An obvious conflict results between these two criteria. Increasing mutation rates to induce exploration for better solutions slows the rate of convergence and required additional computation time. The results of this analysis are shown for the TRIPLEX and ASYMMETRIC LAMP problems in Figure 7-1. Each criterion has been individually normalized so that zero on one

criterion denotes the lowest criterion value attained from the analysis (one denotes the highest value of the analysis), but does not equal a zero on the other criterion. Note therefore that the criterion average shows a normalized, general average of the two criteria. The legend at right shows the corresponding mutation probabilities on the lower axis of the figure.

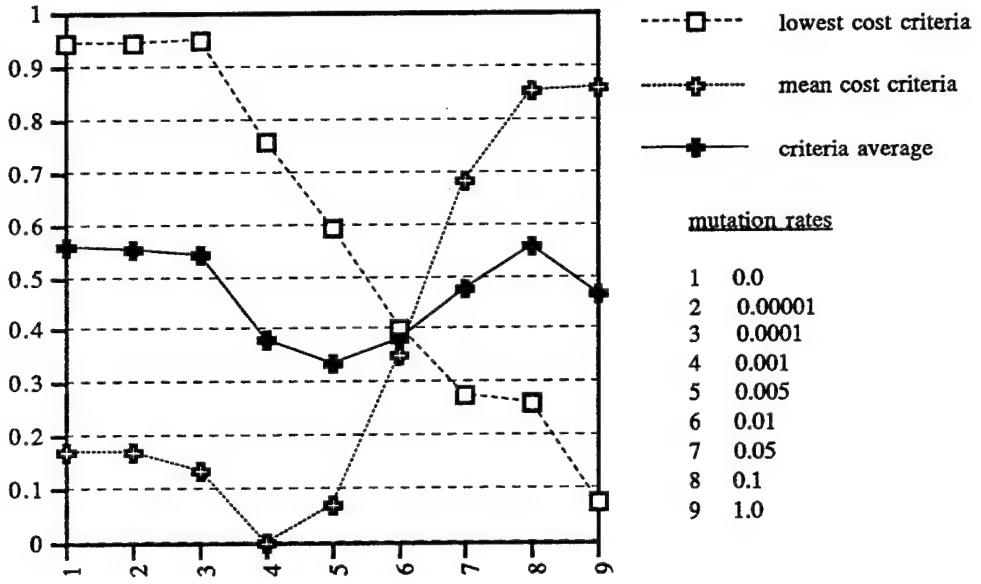


Figure 7-1: Mutation rate comparison

As suspected, the best individual solutions of the TRIPLEX and ASYMMETRIC LAMP problems are found by injecting the highest rates of mutation into the ga. This is shown by the continual improvement of the lowest cost criterion as mutation is increased to 100%. "Saturating" the population with diversity throughout the run augments exploration to allow a wide search of the design space. However, too much population diversity reduces the ga's ability to effectively sort through the population to isolate schemata of high fitness and hinders the confidence in the solution obtained.

Figure 7-1 shows that very high mutation rates prevent the population from converging and concentrating its search. This observation is especially applicable to complex fault tolerant system designs because computational constraints prevent steady-state operation from being reached. Therefore, the DM wants to know the reliability of the solution presented after the algorithm has been permitted a reasonable attempt at the problem. Figure 7-1 shows that this assurance is best met at mutation rates of 0.001 to 0.005.

Nevertheless, a tradeoff must be made between the two criteria to allow

for good exploration with good solution reliability. The average line shows the tradeoff when the two criteria are weighted equally. Using this as the basis, we conclude that mutation rates in the range of 0.001 to 0.01 should be used for fault tolerant system design to maximize the ga's robustness. This range coincides with the suggested rates of 0.005 to 0.01 by Schaffer, et. al., 0.001 by De Jong, and 0.01 by Grefenstette [17].

Mutation in this thesis is considered in the common convention of a rate that affects individual bits. Consequently, the number of mutations expected to occur in the population is a function of the size of the population and the number of bits in each string. The relationship between these parameters and mutation rate is not explored rigorously in this thesis, but some thoughts on the subject follow.

In any population, the total number of bits (N) is the product of its size (P) and its binary string length (l):

$$N = P * l \tag{1}$$

For example: 10 strings * length 10 = 20 strings * length 5 = 100 bits. Specifying a mutation rate of 0.01 causes one mutation to occur, on average, in each of these populations. The issue at hand is whether the "dimensions" of the population should affect the mutation rate selection.

The intent of mutation is to inject diversity into a population in hopes of generating new or previously lost schema of high fitness. The schema lie along the string length, while the population size holds different schema combinations. For a fixed number of bits in a population, therefore, a larger population size holds a greater diversity for fewer possible schema (due to the shorter string length). As a consequence, the mutation rate may be lower for larger populations and higher for smaller populations.

But what if the population size is fixed and the length of the string is varied? If mutation is to inject diversity at bit locations to affect schema, then the mutation rate as a "per bit" formulation should account for the number of schemata in each string. As string length increases, mutation rate should increase to affect the larger number of schemata. The string lengths examined in this thesis fell within a relatively narrow range from 10 to 51 bits, and the effect of the string length on mutation performance is not analyzed.

As a result of the above reasoning and the empirical results of this

section, the remaining analyses use a mutation rate of 0.2/P. For the size of the strings involved ($N \leq 51$), this relationship lowers the mutation rate for larger populations and keeps the rate within the recommended bounds, though it ignores the schemata differences between problems.

8.0 Population Size Analysis

Besides mutation rate, the other ga parameter chosen for detailed investigation is population size (P). To date, there has been no reliable rule-of-thumb developed by which P can be chosen by the decision maker (DM) without extensive problem specific/algorithm specific knowledge. A few others have analyzed this parameter in the past, but the tests have been done with a tga only, and the results have varied. The steady-state genetic algorithm exhibits marked differences from the tga in its exploration and convergence abilities and therefore needs its own population size guidelines.

De Jong made the first attempt at generating a rule-of-thumb for P in 1975. He concluded that tga population sizes between 50 and 100 produce robust results. His efforts were contrasted by Grefenstette in 1986, who proposed a tga population size of 30 and growth of the crossover and mutation rates to maintain population diversity. The latest testing was performed by Schaffer, Caruana, Eshelman, and Das in 1991. This extensive tga test determined that population size should be held to 20-30 strings for good average cost performance [17].

8.1 Population size rules-of-thumb

The first analytical attempt known to this author at determining population size was conducted by Goldberg in 1985. He derived an expression for optimal population size based on the expected number of new schemata per population member [17]. This expression can be approximated as $P = 1.65*2^{0.21*length}$, and advocates population sizes of 30, 557, and 3,460,300 for

string lengths of 20, 40, and 100. Obviously, this method suggests excessively large population sizes as the string length (corresponding in most cases to problem complexity) increases.

Vander Velde [19] suggested a different approach that relies on the stochastic nature of the genetic algorithm. He proposed the population size be chosen to correspond to a given "wrong bit" probability at any locus (bit location). This "wrong bit" probability, P_S, is defined as the probability that all members of the initial population have one or more bit locations for which the bits of all members of the population are the same *and* that bit is "wrong". "Wrong" here means that a bit, when mapped from the string into the cost function, cannot contribute to a good solution (optimum). When this occurs, the entire population lacks information essential to the generation of a good solution. The P_S rule-of-thumb below generates a population size that keeps the probability of this undesirable occurrence suitably low.

$$P_s = 1-\left[1-\left(\frac{1}{2}\right)^P\right]^{length} \tag{1}$$

The following table shows the required population sizes for various P_S probabilities.

	number of bits in each string			
P_S	20	40	80	320
1e-4	18	19	20	22
1e-8	31	32	33	35
1e-12	44	45	46	48

Table 8-1: P_S population sizes for various binary string lengths

Note that there is a very small effect of string length on population size for a given P_S. Complex problems (320 bits) require a population size only four members larger than a problem with strings of length 20. A value between 1e-4 and 1e-8 would support the work of Grefenstette and Schaffer, et al., while probabilities lower than 1e-12 would be needed for De Jong's findings. The rule-of-thumb developed by Goldberg is not supported by this relationship because the probabilities required to correlate that large of population sizes are insignificant.

The idea of using "wrong" bit probabilities does not provide the rule-of-thumb for P we desire, but it does show that injecting diversity into a population is easy, which helps justify the use of low mutation rates. In addition, the amount of diversity in a problem is not greatly affected by the number of bits in the strings, which also justifies our intentional negligence of string length by using mutation rates as 0.2/P.

8.2 Results

Population sizes are tested in the range of 10 to 600. The test is performed using both the tga and ssga on the TRIPLEX problem, the ASYMMETRIC LAMP problem, and the TISS problem (see their descriptions in Chapter 2). For this analysis, mutation rate is set to 0.2/P and crossover is held at 0.80. Twenty (20) seeds (runs) are performed for each population size to reduce the result's sensitivity to the random number generator seed.

Twenty Branch and Bound solutions have been obtained from randomly generated starting points and twenty Monte Carlo simulations are performed for each problem. The results of these runs determine the competitiveness required of the ga methods. All ga runs are limited to a maximum number of cost function evaluations to maintain the efficiency pressure established by the Branch and Bound and Monte Carlo methods. If excessive diversity is lost from the population, defined by the product of the bit likeness (PBL) (see Chapter 6), the runs are terminated early.

First of all, an *exhaustive* search of the entire design space has been performed for the TRIPLEX and ASYMMETRIC LAMP problems. Cost function values are normalized such that 0 refers to the absolute known minimum for the problem and 100 refers to the average results obtained by the twenty Monte Carlo simulations for each problem. A Monte Carlo simulation of 400 points was used for the TRIPLEX problem, a 400 point simulation was used for the ASYMMETRIC LAMP problem, and a 10,000 point simulation was used for the TISS problem.

A comparison to the Branch and Bound method is desired, so twenty runs are performed with twenty randomly selected starting points. The mean normalized solution is 688.49 for the TRIPLEX problem, 43.89 for the ASYMMETRIC LAMP problem, and 60.20 for the TISS problem. Understandably, the results are best for the mixed continuous/discrete

ASYMMETRIC LAMP problem because Branch and Bound uses an underlying continuous search algorithm—the greater the percentage of the problem that is continuous, the better Branch and Bound should perform. The ga, on the other hand, approaches continuous problems as finely divided discrete problems, so that performance is maximized on purely discrete problems. The poor TRIPLEX performance for the Branch and Bound indicates that the method can easily stick in local optima. The Monte Carlo simulations used to set the upper cost normalization value fair very well on the TRIPLEX problem because in 400 points they explore 40% of the 1,000 point design space formed by 10 point discretization of three parameters.

The other variable of comparison for the Branch and Bound and ga methods is computational effort. Branch and Bound does not have solution-on-demand capability—it must be allowed to work to its conclusion. On the twenty runs for the TRIPLEX and ASYMMETRIC LAMP problems, Branch and Bound performs an average of 316 and 195 cfe, respectively, to generate a solution. Likewise, the average number of cost function evaluations for the TISS problem is 38,700, with a maximum limit set at 50,000 cfe. Based on these results, the goal is to prove that the ga is able to systematically generate solutions equal to or better than the Monte Carlo simulation in the TRIPLEX problem and equal to or better than Branch and Bound for the ASYMMETRIC LAMP and TISS problems, in less time than Branch and Bound requires to obtain the solutions.

8.3 Traditional Genetic Algorithm Population Size

The first population size analysis is performed on the tga to compare it to the other methods. Figure 8-1 shows the tga performance at each population size for the three test problems. For the TRIPLEX and ASYMMETRIC LAMP problems, the tga is set to run for 5,000 cfe or a PBL of 0.1, whichever occurs first, while termination is contingent upon 50,000 cfe or a PBL of 0.2 for the TISS problem. Notice also that larger population sizes are tried for the TISS problem to account for its greater problem complexity.

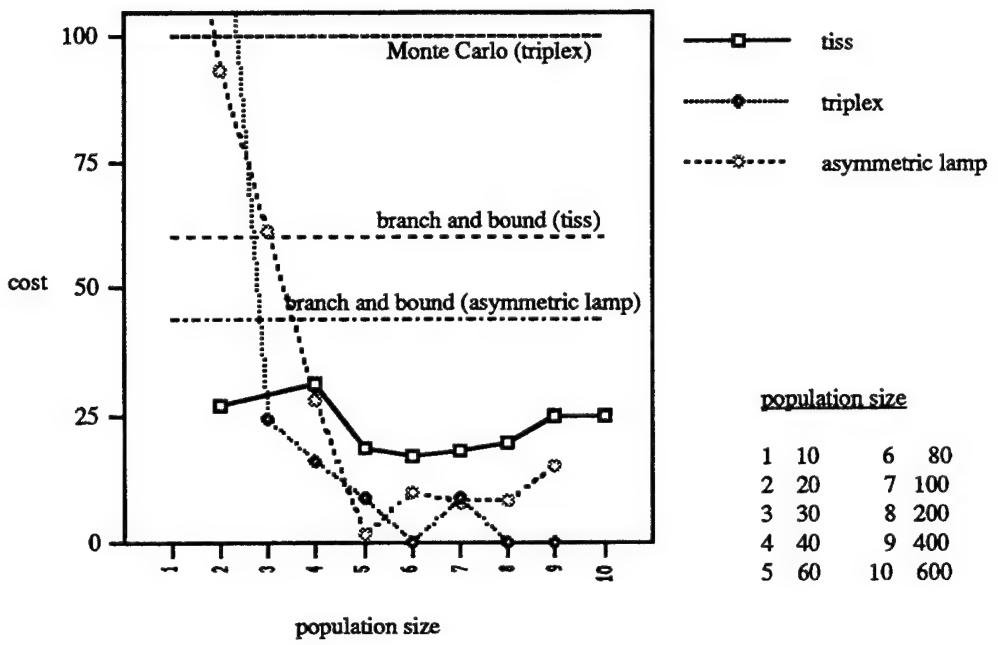


Figure 8-1: Population size comparison for tga

A noticeable difference exists between the problems. The results are generally better as P increases, but the algorithm has greater difficulty with the mixed continuous/discrete problems than it does with the purely discrete TRIPLEX problem. The performance for population sizes below 30 is very poor for the two smaller problems. According to this figure, the optimal population size for a problem similar to the complex TISS problem lies somewhere between 40 and 400 for the tga. The ASYMMETRIC LAMP problem works well with a population size of about 60, while the simple TRIPLEX problem obtains good performance for population sizes greater than 60. This figure also shows the method that generated the most competitive results for each of the three problems. It can be seen that the tga outperforms the Branch and Bound and Monte Carlo methods (on the basis of answer quality alone) for $P > 30$.

As stated earlier, the Branch and Bound and Monte Carlo methods are able to generate solutions reasonably fast. In order to compare the genetic algorithm properly, the tga is stopped at a number of cost function evaluations competitive to the other two methods for each problem. As a consequence, Figure 8-2 shows the tga performance as a function of the same population sizes, but when constrained to 400 cfe for the two smaller problems. The TISS problem is not included because neither of the other methods provided any

worthy competition to the tga! For that problem, Branch and Bound obtains a solution of 60.2 with 38,700 cfe, while the Monte Carlo method generates solutions of 242 and 100 for simulations of 1,000 and 10,000 cfe respectively.

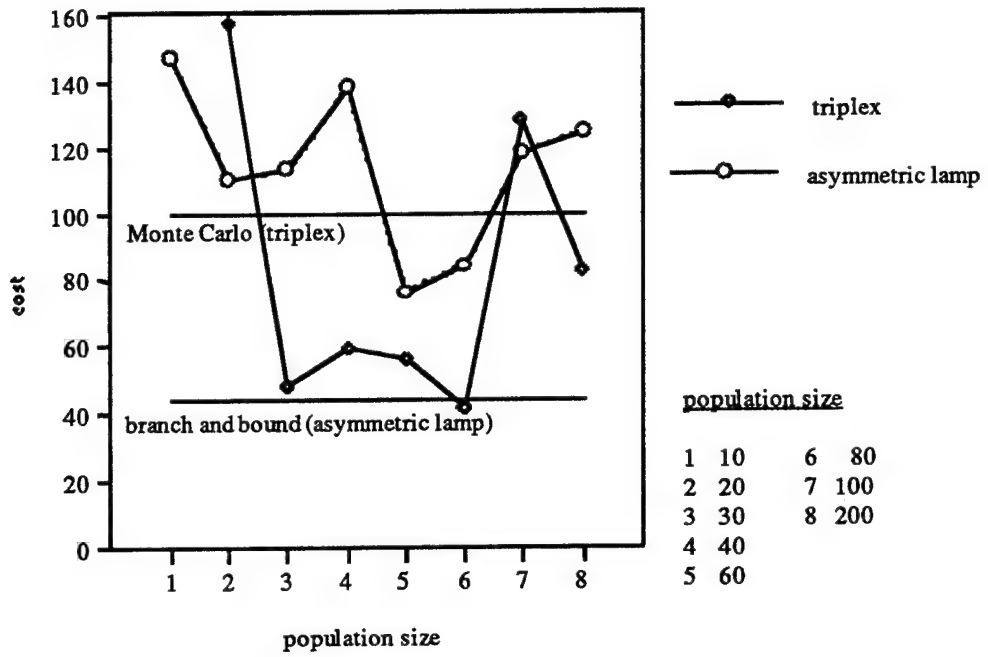


Figure 8-2: tga population size analysis for competitive runs

This figure shows that the competitiveness of the tga is limited in mixed parameter and/or small problems. The best performance of the tga occurs with a population size of 60 (exact correspondence to the optimal value in Figure 8-1), but when limited to 400 cfe its solution is 75.8 compared to the Branch and Bound solution of 43.89.

The tga performance on the fully discrete TRIPLEX problem is better than that of the Branch and Bound and the 400 cfe Monte Carlo attempts. The best tga performance for this small problem was realized for population sizes of 30 to 80 members. From this analysis, we conclude that the optimal P is 60 members for a tga performing fault tolerant system design.

8.4 Steady-State Genetic Algorithm Population Size

The ssga performance at the ten attempted population sizes begins to show its superiority over the tga. Figure 8-3 reveals that ssga performance exceeds that of the Branch and Bound as P increases. This analysis uses the same termination criteria indicated for Figure 8-1: 5,000 cfe/0.1PBL for the two smaller problems and 50,000 cfe/0.2 PBL for the TISS problem. The results of the test problems are also very consistent with one another, which shows that the ssga is less affected by the continuous portions of the ASYMMETRIC LAMP problem than the tga is. There appears to be a monotonic increase in the solution quality with increased population size.

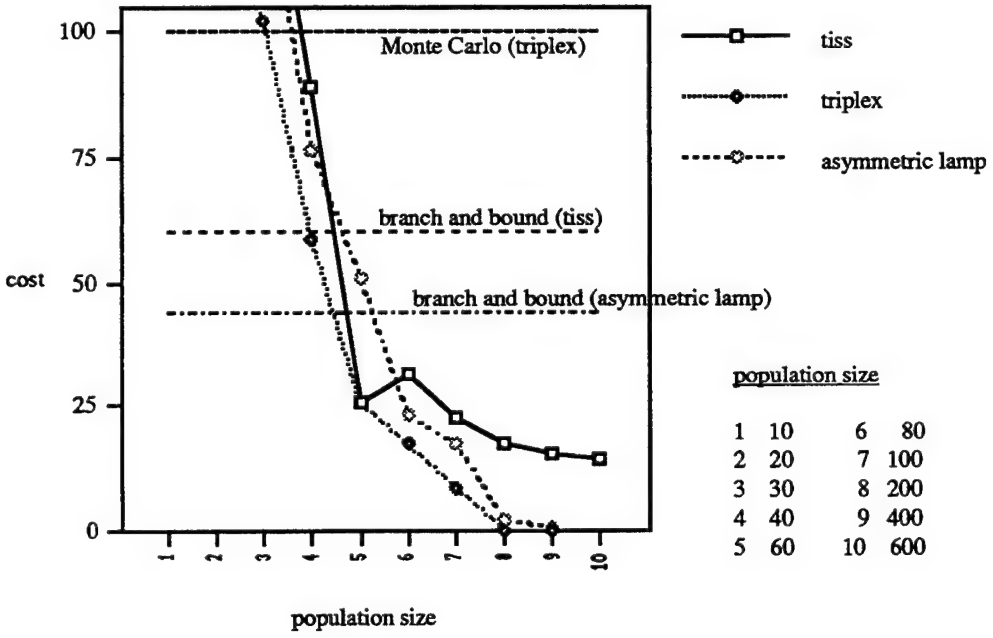


Figure 8-3: Population size comparison for ssga

The ssga requires much fewer cost function evaluations than the tga. The number of cfe needed to reach termination on the basis of PBL increases quadratically with population size and only exceeds *half* of the maximum allowed cfe in one instance (P = 400 for the ASYMMETRIC LAMP problem). Any population size greater than 40 is superior for the discrete problems, while a slightly larger P of 80 allows the mixed problem performance of the ssga to exceed that of its competitors. Again, remember Figure 8-3 shows a comparison of solution quality only. The impact of time constraints such as

limitations of time or computational capability must also be addressed as shown in Figure 8-4.

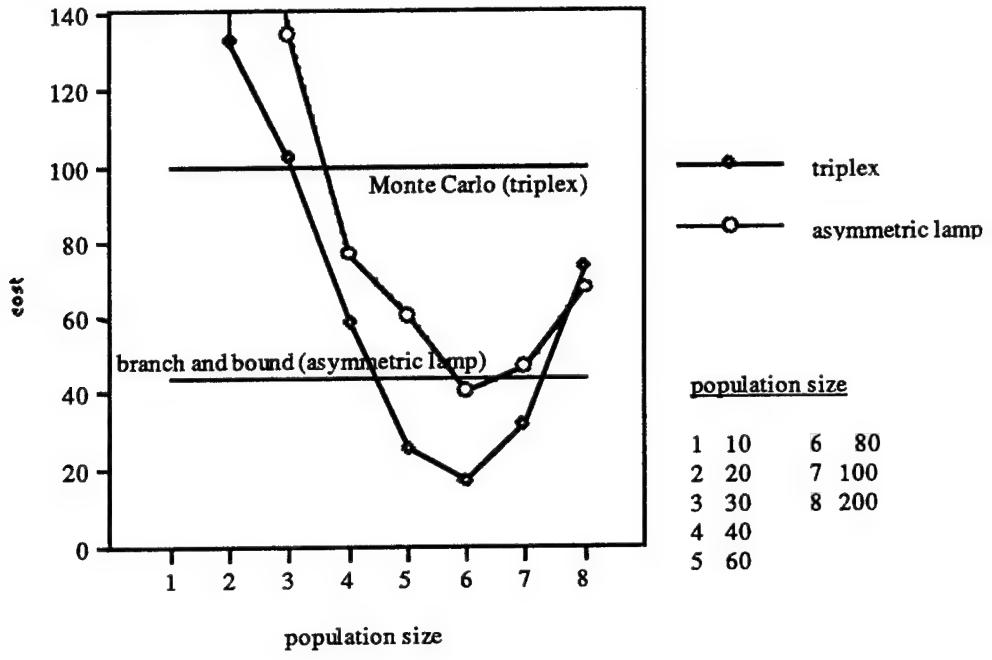


Figure 8-4: ssga population size for competitive timed runs

The ssga performance on the TRIPLEX problem is again superior. The ssga is able to outperform the tga, Branch and Bound, and Monte Carlo methods for all population sizes above 30 members. The figure shows that the best performance can be achieved by using a population size of about 80. Values far from this value show decreased short-run performance. The decrease in solution quality with population size increases above 80 on the time constrained problems is most likely due to the lack of sufficient time for the ga operators to effectively sort through the population and focus their efforts on high fitness areas of the design space. This behavior should be noted by the decision maker (DM), who must determine the time constraint placed on the algorithm—the looser the time constraint, the larger P can be made and the better the solution quality that can be obtained.

The performance of the ssga on the ASYMMETRIC LAMP problem is better than that of the tga, and is very competitive with the Branch and Bound method when the proper P is used. Figure 8-4 shows that a population size between 80 and 100 makes the ssga competitive even though the problem is partially continuous and therefore more favorable to Branch and Bound.

8.5 "Wasted" Binary Coding

As discussed in Section 3.3, choosing to use a binary alphabet for parameter coding introduces "wasted" space on the coded string. Section 3.3 uses the coding of the first 5 integers as an example. This requires a binary string of l = 3, which also has room for integers 6, 7, and 8. A crossover between 3 [0 1 1], and 4 [1 0 0] between the second and third loci (counting from the right) produces 8 [1 1 1], which is not in the parameter space. In this thesis, disallowed parameters are reverted to the value previously assigned to them—all other string changes created by reproduction are retained.

The TISS problem has 17 parameters with 5 discrete bin choices each. As such, each parameter coding, like the example above, has 3 disallowed bit combinations. In the entire string, 3*17 = 51 disallowed parameter combinations exist compared to the 5*17 = 85 allowed. In other words, 3/8 = 37.5% of each string is a disallowed combination. Reverting the affected parameters maintains convergence, but effort is still expended for each appearance, slowing down the method. The effort of keeping disallowed combinations from appearing in the population is small in most cases, but when coupled with the "wasted" crossover and/or mutation operations that produced the disallowed combination, the effect may be significant in some problems.

The frequency of this occurrence is shown in Table 8-2 for both ga methods used on the TRIPLEX and TISS problems. The values in the table are percentages of the total cost function evaluations that required at least one disallowed parameter to be reverted. Remember that a cost function evaluation occurs whenever crossover or mutation is performed on a string.

A Monte Carlo operator on either problem would produce a disallowed value 37.5% of the time. Therefore, the 3% to 10% disallowed generated by both ga methods is three to ten times better in limiting the effort to the proper domain. The reason for the difference in wasted effort for the different problems is not easily determined. It is likely due to beneficial schemata in the TISS problem that appear on disallowed strings. The ga thus looks often in disallowed regions while trying to explore highly fit schemata.

	TRIPLEX problem		TISS problem	
P	tga	ssga	tga	ssga
10	7.38	7.50		
20	5.26	5.08	16.08	16.46
40	4.10	4.70	9.66	10.89
60	3.46	4.65	7.23	9.45
80	3.33	4.41	5.90	9.49
100	3.07	4.43	5.21	8.08
200	4.75	3.97	3.79	7.35
400	5.67	3.92	4.50	6.98
600			4.82	7.03

Table 8-2: "Wasted" effort caused by binary coding

This also table shows that for some reason the ssga generally creates more members with disallowed portions.

Table 8-2 shows that the ga not only produces robust, efficient results, it also effectively limits its operation to beneficial areas of the coded design space and is not greatly influenced by "wasted" space of binary coded parameters. In addition, Table 8-2 proves that the ga user can confidently choose P anywhere within the wide ranges covered by this analysis without hindering the ga's ability to deal effectively with defined regions of the string.

9.0 Summary of Single Criterion Design

The design of fault tolerant systems relies heavily on (1) the availability, accuracy and completeness of a system model, (2) a systematic and efficient design approach capable of handling large component numbers, and (3) an optimization algorithm that adapts readily to model changes and reaches a satisfactory solution in a reasonable amount of time. This thesis has shown that the combination of the Markov Modeling Method, the Optimal Design Process, and genetic algorithms is an effective combination for dealing with real-world fault tolerant system design. Early work at the Charles Stark Draper Laboratory developed the interactions of the Markov Modeling Method and the Optimal Design Process in a continuous problem framework [8]. It has been the intention of this thesis to expand that work to include the robustness of the genetic algorithm (ga) and its ability to optimize the discrete parameter problems that often appear in fault tolerant system design.

The genetic algorithm readily changes to model expansion and additional problem complexity because it never needs to know details of the system it optimizes. *The cost function and the associated parameter-to-string mapping are the only parts of the genetic algorithm method that change in response to system alterations.*

The genetic algorithm works on simple principles that parallel natural biological systems. It uses random choice in a directed search process for optimization. The three key operators that dictate ga performance are reproduction, crossover, and mutation. These operators evolve a population of potential solutions to increase the average solution quality of the population.

The ability of an optimization method to deal effectively with constraints is crucial to fault tolerant system design. The most common constraint in such problems is parameter constraints, where the design variables are limited to a specified range or a discrete set of choices. The genetic algorithm deals with this issue directly by operating directly and solely on the domain allowed through the use of parameter coding. Function constraints, on the other hand, are generally dealt with in ga applications through the use of a penalty function applied directly to the cost function *problem-specifically* by the user. This thesis introduces a new approach that relies on the constraint space information inherently contained in the ga population to automatically assign the penalty function. A multiplicative penalty function applied to the fitness scaling of strings that violate constraints is included to illustrate the approach.

Two types of single criterion genetic algorithms have been explored: the traditional genetic algorithm (tga) evolves whole "generations" of potential solutions in tandem; while the steady-state genetic algorithm (ssga) has been developed by the author to enhance the convergence and speed of the ga by eliminating generational replacement. This thesis has shown that the ssga has better convergence characteristics than the tga in fault tolerant system design. The termination criterion "probability of bit likeness" (PBL) has been developed to capitalize on the convergence characteristics of the ssga and to allow termination at a prescribed level of population diversity loss.

The proper implementation of the mutation operator rate and the population size have been explored for several fault tolerant system design problems.. The crossover rate has not been examined due to a general agreement on its implementation by most ga literature. The effect of mutation is shown to be positive, and the proper mutation rate was found to coincide well with the results of other ga parameter studies.

Mutation rates of 0.001 to 0.01 provided the best performance on the three test problems examined. To coincide with the recent understanding of the ga community that population size and string length affect optimal mutation performance, however, the mutation rate recommended by this thesis is 0.2/P for small string lengths. Longer string lengths (>> 50 bits) were not examined by this thesis, but should generally account for the larger number of schemata in longer string lengths by increasing the mutation rate.

To date, the existence of a reliable rule-of-thumb for determining population size (P) has not been produced. This thesis explores determining population size using the string length and the probability of losing critical levels of diversity from the string (P_s). This idea does not provide a reliable rule-of-thumb, but it does show that injecting diversity into population is very easy, so that the combination of properly sized populations and adequate mutation can easily provide the necessary ga population diversity.

The optimal population size for the ga when optimizing fault tolerant system designs has been explored empirically and has been determined to be a function of the type of ga used and the computational constraints imposed.

The tga performance is optimal when population size is between 60 and 80 members. This conclusion is based on the consistent performance for all three test problems, representing various string lengths and problem complexities. The computational effort expended by the tga increases linearly with additional population members, such that P = 60 usually requires less computational effort than P = 80.

This thesis finds that the ssga exhibits excellent performance over a wide range of population sizes. If not severely constrained by limited computational effort, the ssga can be expected to provide exceptional performance for $P \geq 60$. On large, complex problems like the TISS problem, very large populations of 600 members produce the best assurance of a quality solution. When time constraints are applied, though, the ssga still exhibits superior performance, but like the tga, population size should be limited to the range 60-100.

The computational effort of the ssga goes up quadratically with increases in population size. Though the ssga appears to be very robust and the decision maker (DM) may choose any population size in the above range without concern, the more knowledgeable user may wish to adjust the population size to best suit her wishes: for more accuracy by increasing population size to 100, or more efficiency by decreasing population size to 60. As population size is approximately doubled from 60 to 100, the effort is approximately squared. This general relationship holds for all population sizes tried on the ssga.

Even at the tga's optimal population size of 60 where the average solution outperforms the ssga of the same population size, three times more tga computational effort is required to achieve the solution. Therefore, it is the

general conclusion of this thesis that the ssga is superior to the tga for optimizing fault tolerant system design problems.

One of the more important conclusions of this thesis is that the genetic algorithm is superior to the Branch and Bound method. The ga is a much more robust and efficient method for optimizing the design of fault-tolerant systems as shown by Figure 8-1 through Figure 8-4. In contrast to the ssga with a population size of 600 on the TISS problem, the Branch and Bound delivers solutions 4 times worse and performs 2.8 times the computational effort.

The ga has solution-on-demand capability. Following a reasonably brief operating period, the ga is capable of delivering a solution to the DM. Further efforts may improve that solution, but the DM need not wait for the algorithm to complete its operation. Branch and Bound, on the other hand, is usually incapable of finding a quality intermediate answer. In a small test of this assertion, only 3 of 10 Branch and Bound trials on the TISS problem generated *any* solution after 4,000 cfe, and those three were very poor.

By direct comparison of the genetic algorithm and other discrete optimization results, such as the Monte Carlo method and the Branch and Bound method, the genetic algorithm is shown to provide a very efficient and robust approach to discrete optimization. Because the system model requires no derivative information, the genetic algorithm formulation proves to be much easier to develop. Second, the genetic algorithm need not be changed with the system's variations because the cost function and its associated string-to-parameter mapping are the only parts of this method that change in response to system alterations. Third, since the genetic algorithm searches from a population of points, it converges to an acceptable solution faster than methods that move along a single optimization path and is capable of providing solutions to the DM at any point of its operation. Even with a large population size of 600, the ssga only requires 13,580 cfe to converge and generates a solution 75% better than Branch and Bound is capable of generating in 38,700 cfe. Finally, a concern for any optimization technique is the reliability of the result. This thesis proves that the genetic algorithm can be relied on to provide a suitable solution to complex, integrated fault-tolerant systems while maintaining its model simplicity and efficiency attributes.

10.0 Multicriteria Optimization

10.1 Background

Multicriteria optimization refers to the process of dealing with multiple criteria that a decision maker (DM) wishes to optimize simultaneously. The issue of optimality in this context differs due to the often conflicting relationship between criteria. The general single criterion problem with n design variables and m constraints is:

$$\begin{aligned} &\text{Optimize:} && J(\mathbf{x}); && \mathbf{x} = [x_1, x_2, \ldots, x_n]^T \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; && i = 1, 2, \ldots, m \end{aligned} \tag{1}$$

The general multicriteria optimization problem with n design variables, m constraints, and l criteria is [2]:

$$\begin{aligned} &\text{Optimize:} && \mathbf{J}(\mathbf{x}) = [J_1(\mathbf{x}), J_2(\mathbf{x}), \ldots, J_l(\mathbf{x})]^T; && \mathbf{x} = [x_1, x_2, \ldots, x_n]^T \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; && i = 1, 2, \ldots, m \end{aligned} \tag{2}$$

The vector **J** is the multicriteria objective function formulation with elements J_1, J_2,..., J_l as the individual criterion objective functions. It is important to note that the individual criterion functions are merely listed together in a vector; they are not added, multiplied or otherwise combined in the general multicriteria problem.

Single criterion optimization seeks to identify an optimal solution. This is defined as simply the feasible solution (or solutions) that gives the best

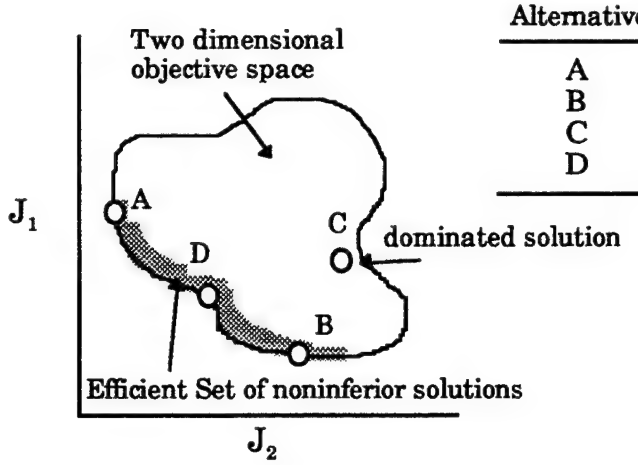
objective function value. The desired location in the design space is unique even if alternative optima exist.

The single criterion notion of optimality does not work in the multicriteria context, however. A new concept involving efficiency (also called domination) serves a similar function in multicriteria problems. The desire of the analyst in multicriteria problems is to assist the DM in isolating a preferable solution that lies in the set of efficient points (also called the dominant or Pareto dominant set). The means for accomplishing this goal is explained later—first we must assemble the basics of the multicriteria problem.

The notion of efficiency (noninferiority, nondominance, or Pareto optimality) is best illustrated by way of example. Suppose four solutions exist for comparison in a two criteria minimization problem [2] (see Figure 10-1). Alternative C is *inferior* to B and D because it has criteria values worse than the others in both criteria in each case. Furthermore, B and D dominate C because their values are better than C in both criteria. Alternative A does not have a dominance relation with any of the given alternatives B, C or D because the J_2 value of A is better than all the others while its J_1 value is worse than all the others. Alternatives B and D do not have a dominance relationship either for the same reason. Solutions that are not dominated by any part of the objective space are called *efficient*, such as A, B, and D of Figure 10-1, because as defined in Cohon:

"A feasible solution to a multiobjective [multicriteria] programming problem is non-inferior [efficient] if there exists no other feasible solution that will yield an improvement in one objective without causing a degradation in at least one other objective." [2]

Figure 10-1 shows efficiency in a J_1 - J_2 plane. The objective space shown must be distinguished from the decision space containing the feasible design variable (parameter) regions. All interior solutions of the objective space such as alternative C are inferior no matter if the individual criteria are to be minimized or maximized. In Figure 10-1, both criteria are to be minimized, so that the optimal value would lie at the origin if the entire quadrant were feasible.



Alternative	J_1	J_2
A	10	10
B	3	80
C	7	90
D	5	30

Figure 10-1: Two dimensional multicriteria objective space

The general rule for determining efficiency in a minimization problem is that when no feasible solutions lie in a "cone" to the lower left of a point of interest—that point is efficient. Conversely, any point that has some portion of the feasible objective space in a "cone" to its lower left is dominated. The shaded region of Figure 10-1 shows the full efficient set of the problem.

Multicriteria methods attempt to allow the analyst and decision maker (DM) to perform the aspects of system design they are best prepared to deal with. The analyst represents the design engineer with a great deal of technical knowledge pertaining to the multicriteria fault tolerant design process and the optimization methods employed. The DM, on the other hand, represents the individual or group that must make the critical design decisions when presented with the tradeoffs between conflicting criteria. The DM need not have any technical knowledge concerning optimization for multicriteria design to be effective. Multicriteria optimization for fault tolerant system design couples (1) the analysis of potential solution dominance (solutions should lie in the efficient set) and (2) the more intuitive, empirical, and "gut level" decisions that must be made between alternatives in the efficient set to arrive at a satisfactory system design solution.

10.2 Multicriteria Technique Classification

Classification of the various multicriteria optimization techniques is based on the interaction of the method, analyst (who controls the method), and the DM (who chooses the solution). Techniques are classified in many ways by different authors, but for the context of this thesis multicriteria techniques include four main types: no preference articulation, a priori articulation, progressive articulation, and a posteriori articulation. The techniques that fall into these categories will not be explained in detail, but the general approach of each category will be explained to illustrate how this thesis fits into the broader framework of multicriteria design.

10.2.1 No preference articulation

This class of multicriteria optimization requires no DM interaction with the method. Commonly referred to as the Global Criterion Method, in this method the analyst applies an optimization technique to provide the DM with a single solution from the efficient set. The solution comes from a two part problem:

1) Solve the problem for individual criteria:

$$\begin{aligned} &\text{Optimize:} && J_k(\mathbf{x}); && k = 1,2,...l \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; && i = 1,2,...,m \end{aligned} \tag{3}$$

The solutions to this step are designated $J_k(\mathbf{x}^*)$

2) The second step is a goal minimization:

$$\begin{aligned} &\text{Minimize:} && \min \sum_{k=1}^{l} \left| \frac{J_k(\mathbf{x}^*) - J_k(\mathbf{x})}{J_k(\mathbf{x}^*)} \right|^p && p \geq 1 \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; && i = 1,2,...,m \end{aligned} \tag{4}$$

The optimal solution depends on the choice of the norm p. The solution is the point of the objective space where the tradeoffs between the criteria are the smallest for the given norm p. The solution produced is in the efficient set if all criteria are strictly positive so that measurements from the objective space origin (where $J_i = 0; i = 1,2,...,l$) to $J(\mathbf{x}^*)$ are always positive distance quantities

[2]. This method would produce a solution at the "knee" of Figure 10-2 where the choice of solutions is relatively easy.

This method by itself, however, does not provide adequate information about the criteria tradeoffs to allow the DM to be confident in the solution. No justification of the solution is provided and the arbitrary choice of p affects the solution obtained. This type of analysis is a "no brainer" that is often avoided in actual multicriteria design.

10.2.2 A priori articulation of preferences

Several methods fit this category. All of them require the DM to have a good grasp of what she expects to see from the analysis and what she would prefer to do with that information *before any analysis takes place*. The way preferences are articulated identifies the individual methods. The utility function method, lexicographic method, surrogate worth tradeoff method, and goal programming method all require a priori preference articulation.

These methods place a relatively large demand on the DM in terms of the information required. They have the major disadvantage of requiring the DM to spell out very explicitly her preferences. This process is usually very time consuming, which makes DM involvement difficult since no feedback can be immediately returned. Also, the DM may not know enough about the problem to effectively state preferences. The unknowns of feasibility and the tradeoffs necessary to achieve stated preferences often make DM reluctant to express their preferences.

10.2.3 Progressive articulation (interactive programming)

Interactive methods use DM preferences to perform local searches of the multicriteria design space. Progress toward a solution of the problem is accomplished by iterative interaction of the method with the DM, either with or without the assistance of an analyst. As local progress is made to a solution, new reactions of the DM are used to redirect the next local search. These methods have the distinct advantage of often producing solutions that the DM particularly likes because of the constant interaction and the better visibility of where the solution came from.

Methods often placed in this category include: interactive goal programming, the step method, and the sequential multiobjective problem solving technique. These methods have distinct disadvantages, however. First, they require a very high time involvement from the DM . The DM must be present at each iteration of the method to look at intermediate results and express preferences to orient further search. Secondly, there is no guarantee that the solution obtained will be satisfactory to the DM. The expression of local preferences as the method progresses may lead to a solution that is the sum of the satisfactory parts but is not a satisfactory whole.

10.2.4 A posteriori articulation (generating methods).

Finally, there are methods that enlist the assistance of the DM only at very late stages of the analysis. These methods put a comparatively small burden on the DM in terms of the amount of information required.

Using the convention adopted in [2], these methods will be called generating methods for their ability to generate a full or partial representation of the efficient set. Because the analytical analysis has already been performed, the DM need only to react to the results produced. As put in [2], "the emphasis is on the demarcation of the range of choice, not on the explicit definition of preferences." The DM are offered insights into feasibility, necessary sacrifices, and the costs/benefits of each solution of the efficient set. The a posteriori methods inform the DM, not the other way around.

Preference articulation is an overpowering drawback to multicriteria optimization. A priori and iterative methods are usually rendered useless if the DM changes her mind. The very fact that these problems deal with a human-interface with the frequent capacity for indecision or reversal of position lends a great deal of weight to a posteriori methods.

The first three classes of methods have an additional drawback. In them, the role the analyst takes in the process of selecting a problem solution can greatly affect the solution selected and the satisfaction the end-user (DM or otherwise) has with that solution. It is never the role of the analyst to steer the progression toward a solution. If the DM is not well versed in the technical aspects of optimization, especially the complexity of the multicriteria framework, the analyst may have a great deal of difficulty in obtaining the necessary articulation of preferences for the method. In such a case, the

analyst can have a large impact on the progression of the method. Also, such DM will often distrust the solutions obtained because the analyst will have seemed to have pulled solutions "from thin air". These difficulties can usually be overcome, of course, but the premise of this thesis is that the analytical analysis of the problem should be kept as isolated as possible from the expression of preferences.

The analytical methods for approaching a multicriteria problem will always remain short of fully capturing the essence of multicriteria problems. This fact must be appreciated so that the non-analytical articulation of *preference*, an emotional, "gut level", and always individual choice, can be appreciated and allowed to work without the constraints of analytic thought and over-constrained structure.

Generating methods can also be used as a "front-end" approach for other techniques. If the efficient set cannot be analyzed by the DM without additional considerations, articulation of preferences at this point, with a larger picture of the tradeoffs in the objective space available, can narrow the solutions under consideration. Making the efficient set representation produced by the generating method the domain of additional search, the additional computational requirements can be kept to a minimum and the "big picture" context of the narrowed focus can be kept in proper perspective.

The major disadvantage of generating methods is the computational requirement. The fact that these methods provide a full set of potential solutions makes this disadvantage understandable. Many methods in this category require the solution of a complete single-criterion mathematical program for each point of the solution set. For these methods to be worth using, therefore, they must provide competitive computational requirements.

Computationally, the first three categories avoid high costs by full utilization of the preferences given. These methods usually have a great computational advantage over the generating methods, but the high investments of user time weigh heavily against their use.

No one approach is optimum for every multicriteria problem, but this thesis intends to show that a posteriori articulation of preferences has general applicability when coupled with an optimization technique that is rapid, reliable and versatile. Methods in this category include the ε-constraint method, weighting method, and the Multicriteria Genetic Algorithm (MCGA). These methods will be presented in further detail in later sections.

10.3 Multicriteria Test Problems

The fault tolerant design test problems used in this section are essentially identical to those used heretofore. The same fully discrete TRIPLEX and TISS problems are used with the same problem characteristics described in Chapter 2. The only significant difference is that now each problem has more than one criterion to optimize.

Two criteria have been created for each problem, and a third is available if the methods analyzed in the following sections would need to be analyzed for problems with greater dimensions. The three criteria are shown in Table 10-1.

criterion	description	Markov Model dependent?
Purchase Cost	sum of component costs	no
Unavailability	probability of being in failure states	yes
Operating Cost	configuration dependent; accounts for overhead, maintenance, etc.	yes

Table 10-1: Sample fault tolerant design criteria

10.4 Display

The difficulty of multicriteria optimization is not limited to algorithm deficiencies; a DM (often with no technical background) must also use a representation of the efficient set to make a *qualitative* choice of optimality. Assisting the DM choose a single solution through point comparisons or articulation of preferences is contingent upon the quality of the result presentation. The manner in which multicriteria optimization results should be presented depends on the number of criteria in the problem.

In two criteria problems, the best manner of display is a simple plot of the two criteria on perpendicular axes. This display is used throughout this thesis and is easily understood by most DM. The tradeoffs between criteria are easily seen and special interactions of the criteria can make the choice of a single solution much easier. For instance, the points with mild tradeoffs between criteria at a distinct "knee" of the efficient set as in Figure 10-2 usually are good solution choices.

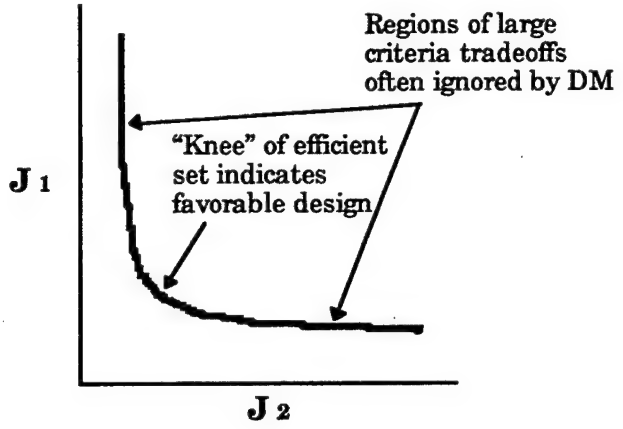


Figure 10-2: Two criteria problem efficient set with "knee"

For problems with three or more criteria, this simple graph is no longer practical. For three criteria and associated efficient sets with appropriate characteristics, 3-D plots, usually with color references, can help the DM to visualize the surface of interest. Another manner to display three criteria is to use a simple graph and display efficient set cross-sections for fixed levels of the third criterion [4]. However, both of these are difficult to construct and only applicable to three criteria problems.

The number of complex, multidimensional points that a DM can directly compare is usually limited to 4 or 5 to avoid information overload at a single step of the analysis. After general regions of the efficient set have been separated, more focused analysis can occur. To facilitate this approach in multiple dimensions, the *profile* device has been established [4]. Figure 10-3 shows an example profile for three points on four criteria. The addition of color and other visualization tools can greatly aid the DM to effectively compare the relative merits of points using the profile.

In a profile, vertical lines indicate the different criteria. The criteria are usually normalized in some region of interest, such that relative values are compared. Points in the design space are designated a particular symbol (circle, triangle, etc.) and their values in each criterion are plotted and connected with lines to show visual continuity. In the minimization problems of this thesis, a dominated solution has all of its symbols above (worse) than at least one other point of the design space.

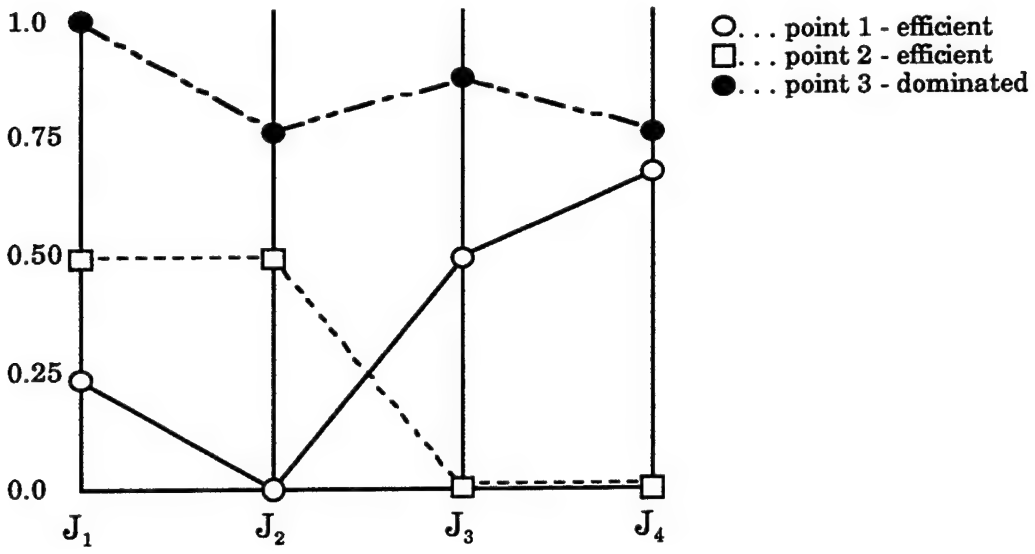


Figure 10-3: Profile display for multicriteria analysis

The profile shows the tradeoffs between potential solutions of the objective space. Comparing a few potential solutions from different regions of the objective space can help the DM determine the amount of give and take required between different regions of the efficient set.

11.0 Multicriteria Generating Techniques

11.1 ε - Constraint Method

The ε - Constraint method, and the Weighting method explained in the next section, both transform the multicriteria problem into a series of one-dimensional (single-criterion) problems for which solutions are obtained in the traditional manner. As with the Branch and Bound method, the ε - Constraint method (from here on simply referred to as the constraint method) is a framework for configuring a difficult problem. It requires an underlying optimization algorithm to solve the stack of sub-problems it creates. The solutions to those sub-problems should map out a representation of the efficient set (**E**).

The framework for this method begins with the general multicriteria problem presented in equation (18) and shown again here:

$$\begin{aligned} &\text{Optimize:} && \mathbf{J}(\mathbf{x}) = \left[J_1(\mathbf{x}), J_2(\mathbf{x}), \ldots, J_l(\mathbf{x})\right]^T; \quad \mathbf{x} = \left[x_1, x_2, \ldots, x_n\right]^T \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; \quad i = 1, 2, \ldots, m \end{aligned} \tag{1}$$

The constrained problem for n parameters, l criteria, and m constraints is:

$$\begin{aligned} &\text{Optimize:} && J_k(\mathbf{x}); \quad k = 1, 2, \ldots, l \\ &\text{Subject to:} && g_i(\mathbf{x}) < 0; \quad i = 1, 2, \ldots, m \\ &\text{and} && J_h(\mathbf{x}) \le L_h; \quad h = 1, 2, \ldots, k-1, k+1, \ldots, l \end{aligned} \tag{2}$$

where the h^{th} criteria are chosen arbitrarily for optimization to create a single criterion problem. The means for obtaining the function constraints L_h is detailed below.

The fundamental idea of this method is that one criterion should be optimized while all others are represented as constraints. This process defines a noninferior solution. Choosing proper L_h to find a desirable number of efficient points, a representation of the efficient set can be generated while *only one criterion has to be optimized* (changing line 1 of equation (2) so that $J_h(\mathbf{x})$ has to be optimized for only one value of k). The manner in which this is done is described graphically for a two criteria problem in Figure 11-1.

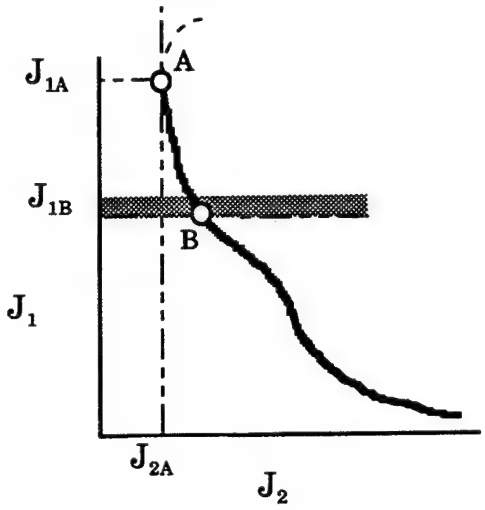


Figure 11-1: ε - constraint multicriteria method

This figure shows two criteria represented as J_1 and J_2. By optimizing on criterion J_2 without placing any condition on the other, the point A is obtained as the minimum. Point A denotes one edge of the efficient set; the other could be found by optimizing on J_1 with no conditions on J_2.

Now suppose that an inequality constraint is placed on J_1 such that J_2 is to be minimized subject to $J_1 < J_{1B}$. The solution to this problem is point B, which is a point on the efficient set. This process can now be repeated to generate the desired resolution of the efficient set. The choice of which criterion is to be optimized and which is to be held as a constraint can be done arbitrarily without impacting the method's performance in most instances, or can be made by the DM based on prior knowledge of the relative ease of optimizing and/or bounding either criterion.

The DM desired resolution is established through the parameter ε. This parameter denotes the change in the constraint bound for one criterion necessary to achieve the desired spacing in other criteria. For example, if the individual J_1 and J_2 problem optima (i.e. the efficient set boundaries) are found to be [100,1] and [200,0.1], we know that J_1 varies from 100 to 200 along the efficient set. If the DM desires a 5 point efficient set resolution, ε is set to (200-100)/(5-1) = 25.

Thus, to provide five point resolution, three points must be located in addition to the two endpoints. With ε = 25, the three necessary constraints are set as:

$$J_{1i} < 100 + i\varepsilon; \qquad i = 1, 2, 3 \tag{3}$$

In more than two dimensions, this method experiences significant drawbacks, however. The difficulty arises that many constraints imposed create infeasible sub-problems. This phenomenon is described in more detail in [4] with a graphical illustration. Though this disadvantage is not catastrophic for the method, it may result in much wasted computational effort because the solutions obtained in such instances will either be a duplication of effort that produces cloned solutions or will not be part of the desired resolution.

The constraint method can be very tedious and computationally expensive. The DM has the option of defining as much or little resolution of the efficient set he or she would like to generate, however, which makes the method reasonably predictable and very parallelizable. Once the individual limits on the criteria are found (which can be done in parallel), the remaining sub-problems needed to create the efficient set are simply a function of those initial solutions as illustrated by equation (3). Every remaining sub-problem can be solved independently. In this way, an analyst with access to highly parallel hardware can make excellent use of his facilities and reduce the otherwise excessive run-time of the constraint method.

In this research, a simple 2-dimensional constraint method framework has been created. The underlying optimization algorithm is the steady-state genetic algorithm (ssga), already proven to be a superior method for the optimization of discrete single criterion fault tolerant system design problems.

As noted earlier, one drawback of the constraint method is infeasibility for constrained design space locations. This disadvantage is dealt with in this

particular implementation by analyzing the initial population of the ssga at each constraint level for feasibility. If the entire initial population is infeasible, the method is instructed to suspect that the current problem is too tightly constrained. To verify this assumption, the initial population is regenerated at random and again checked for total infeasibility. If the second attempt also generates an infeasible first generation, the current constrained problem is discarded. This manner of dealing with infeasible constrained problems requires 2*P cfe to determine the validity of the infeasible first generation assumption. To not include this, check, however, would allow a full ssga run with a fully infeasible population, requiring many cost function evaluations with little chance of finding a feasible, optimal solution.

The intent of this thesis is to accentuate the robustness of the genetic algorithm by maintaining biological analogies whenever possible and keeping the methods' characteristics general by separating problem data from the ga operating domain. In light of this intention, a new formulation of the distance-based penalty function has been developed for this thesis (see Chapter 5) and is included in this implementation of the constraint method.

A multiplicative fitness penalty function (G) is applied to degrade the fitness (F) of ssga strings that violate the imposed bound on J_1, referred to below as g.

$$\text{Fitness:} \qquad F_j' = F_j * G_{ij}; \qquad j = 1, 2, \ldots, P$$

where

$$G_{ij} = \begin{cases} \left(\dfrac{g_i - g_i^*}{\Delta g_i} + 1 \right)^{-3} & \text{if } g_i(\mathbf{x}) > g_i^*(\mathbf{x}); \quad i = 1, 2, \ldots, l-1 \\ 1.0 & \text{otherwise} \end{cases} \tag{4}$$

where P is the population size, j is an index over the string members of the population, and i is an index over the problem constraints. The quantity Δg is the distance between the efficient set boundaries measured along the g (J_1) axis. The inequality constraint is presented as a function of the actual constraint bound to show the effect of crossing the inequality threshold g*.

This G has very favorable effects on the test problems of this thesis. The fitness is degraded as a function of the distance from feasibility. The value of G drops off sufficiently fast to give a reasonable assurance that strings with large constraint violation distances will not be assigned competitively high

fitness values. The decay rate (-3) is chosen arbitrarily for this thesis to provide a penalty of approximately 90% to those strings at the bound of constraint "closeness" (see Chapter 5). Results for the constraint method are discussed in Chapter 15.

11.2 Weighting Method

The weighting method generates a representation of the efficient set systematically by varying a single weighted sum function of the problem criteria. It accomplishes its task somewhat faster than the constraint method. Its major disadvantage is its vulnerability to missing or misrepresenting portions of the efficient set.

The weighting method operates in the following manner to reduce the multicriteria problem of l criteria to a single criterion weighted sum [4]:

$$\begin{aligned} &\text{Optimize:} && \mathbf{J} = \left[J_1(\mathbf{x}), J_2(\mathbf{x}), ..., J_l(\mathbf{x})\right]^T \\ &\text{Subject to:} && \mathbf{g}(\mathbf{x}) < 0 \end{aligned} \tag{5}$$

is the multicriteria problem transformed into:

$$\begin{aligned} &\text{Optimize:} && \mathbf{wJ} = \sum_{k=1}^{l} w_k J_k(\mathbf{x}) \\ &\text{Subject to:} && \mathbf{g}(\mathbf{x}) < 0 \end{aligned} \tag{6}$$

The problem defined in equation (6) defines a point in the efficient set. Systematically repeating the process for various w_k defines some representation of the efficient set.

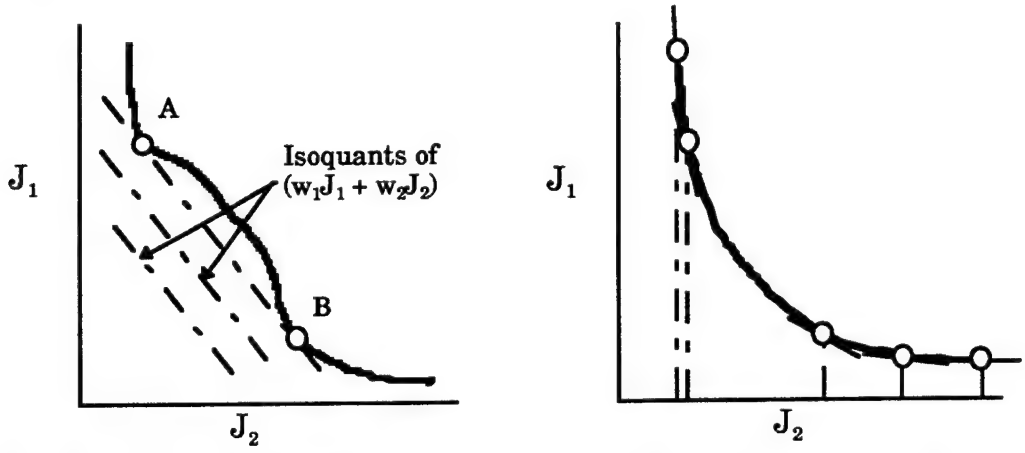
The weights do not have to provide meaningful interpretation for the problem. The weights are a means of mapping out the efficient set and can be considered arbitrary numbers generated simply for the mapping produced or as relative "worth" assigned to each criterion. In fault tolerant system design, however, the DM usually does not know what the relative "worth" of different criteria should be and cannot confidently place weights on criteria to attain optimum solutions. As such, a systematic change of the weights is usually used.

The manner in which the weights w_k are varied depends only on their *relative* values, not their absolute size. With two criteria, for instance, the

weighted sum of $(J_1 + 3J_2)$ produce the same solution on the efficient set as the sum $(3J_1 + 9J_2)$. This relationship leads most applications of the weighting method to use positive relative criteria weights that sum to 1.0. The combination of criteria weights can be mapped into the design space as surfaces of constant "value" called isoquants. In two-dimensional problems, the isoquants are straight lines of constant "value" represented by:

$$C = w_1 J_1 + w_2 J_2 \tag{7}$$

The weighting method searches for points of the feasible objective space that fall tangent to the isoquants and minimize the function value of the isoquant at that point. Figure 11-2a below shows the isoquants for some combination of weights in a two criteria problem. In the particular instance shown, the solution is not unique.



(a) the weighting method may skip reentrant portions like that between A and B.

(b) the weighting method makes control of efficient set resolution difficult.

Figure 11-2: Weighting multicriteria method disadvantages

Comparing the weighting method to the constraint method, the constraint method usually gives a more reliable description of the efficient set because the user directly controls the point spacing by the constraints imposed. The weighting method has the tendency to skip over "reentrant" portions of the efficient set as in Figure 11-2a. The second plot shows that since problem geometry affects where the points are located (constant changes in the weighting increments do not create equal spacing between

solutions), the analyst cannot predict the spacing of the points and may not provide a satisfactory representation to the DM.

Another disadvantage shown in (a) above is that weights may not provide unique solutions. Both A and B above are obtained by the same weight configuration. The particular point found would depend on the underlying single criterion optimization method used and, in the case of gradient based methods, would depend on the method's starting conditions.

The weighting method does have a significant computational advantage over the constraint method. Its single criterion cost function is a simple sum of the criteria functions without regard for constraint difficulties. The weighting method does not waste effort as the constraint method does, looking for solutions in infeasible areas. Unless external constraints are imposed, every weighted combination of criteria is feasible.

The weighting method is not used in this thesis, but is presented due to its popularity, generality, and ease of use. The methods used in this thesis must prove to be competitive to the known strengths and weaknesses of the weighting method to justify their use in fault tolerant system design.

11.3 Multicriteria Genetic Algorithm

The Multicriteria Genetic Algorithm (MCGA) expands upon the basic structure of the steady-state genetic algorithm in its attempt to simultaneously optimize multiple criteria. The MCGA forms a representation of the efficient set through the use of the same three basic ga operators: reproduction, crossover, and mutation. For reproduction, since the concept of fitness no longer applies with more than one criterion, tournament selection is used instead, which places candidate members in competition with each other.

The MCGA creates a "partial order" of the members of the population based on dominance. Dominance replaces fitness for reproduction testing and allows the MCGA to optimize all criteria simultaneously with much less effort than conventional multicriteria generating methods, due to the ga benefit of working from a population of points. This is especially true in the multicriteria context because, unlike the single criterion problems where having a population was necessary to find a single optimum, here the population itself evolves as an entity that *is the solution*.

The first reference to a multicriteria genetic algorithm we were able to locate was in [10]. The algorithm they used was called the Niched Pareto Genetic Algorithm (NPGA), and used domination tournaments for reproduction selection, equivalence class sharing for maintaining the necessary population diversity along the efficient set, and a distinct generational algorithm similar in its basic form to the tga. The NPGA was specifically a proof-of-concept of the capability of genetic algorithms to optimize multicriteria problems. The NPGA was not rigorously tested, and it was analyzed for its on-line performance only. The MCGA of this thesis uses the same framework as the NPGA. It differs only in some of the method parameters available and in its orientation toward fault tolerant system design.

Reproduction

In the analyses of this thesis, reproduction occurs in one of two ways, depending on whether single or multicriteria optimization is performed. The single criterion methodology was described in Section 3.5 and uses selection and fitness testing. These steps determine which individuals of the population are reproduced based on factors equivalent with the environment, mating preferences, and individual strengths. The multicriteria manner of reproduction incorporates *tournament selection*, which allows for direct competition among members of the population without the need of a separate fitness scaling of string worth.

In tournament selection, members of the population directly compete to determine who will survive and reproduce. In any competitive event, the possibility always exists of a tie. This possibility is handled by applying *equivalence class sharing*.

11.3.1 Tournament Selection

A competitive tournament is created from a representative sample of the entire population to reduce the unnecessary computational burden of using the entire population each time a parent is to be chosen. In this implementation, the set of attributes (criteria, phenotypes) for the tournament set is used to create a partial order of member domination [10]. The following paragraphs describe the tournament selection procedure. The process is shown as the "attribute tournament arena" box of Figure 11-5.

Competitors

First, two unique (not clones) candidates for reproduction are selected from the population at random. Next, a set of representative individuals is also selected from the population at random. Reproduction selection is based on the relative domination of the candidate strings to the representative set.

Tournament set size (t_{dom})

The size of the representative set significantly impacts the method. If the set is too small, t_{dom} is not representative, and a realistic domination rank of the candidates cannot be determined. On the other hand, making the set too large wastes valuable computational effort.

Just as the ga operates to continually improve the average fitness of the population, the MCGA operates to continually move the population towards a representation of the actual efficient set (**E**). Consequently, the value of t_{dom} also affects the pressure the method puts on the movement toward **E** and impacts the tradeoff between efficiency and exploration. A smaller t_{dom} means that the candidates will be more likely to show a falsely high dominance, which will reproduce less dominant members at a higher rate—emphasizing exploration over efficiency.

Horn and Nafpliotis conclude the following order of magnitude guidelines for setting t_{dom} for on-line performance:

- t_{dom} ~ 1% of population size (P), too many dominated solutions
- t_{dom} ~ 10% of P, tight and complete distributions formed
- t_{dom} >> 20% of P, premature convergence occurs to small sections of **E**

Definition of a Tie

Once the proper t_{dom} is established, the dominance based selection can also occur in one of two ways to affect efficiency and resolution. First, if dominance is determined when one candidate completely dominates the tournament set, as suggested by [10], the tournament is very selective since both candidates will not be completely dominant in most cases, leading to many ties that have to be decided by sharing—emphasizing spacing and resolution over efficiency of movement toward **E**. Secondly, dominance can be defined as the candidate that dominates the greatest number (or fraction) of the tournament set. In this manner sharing is only required when both candidates dominate the same fraction—emphasizing efficiency over resolution because fewer ties will be declared and less sharing has to be performed.

The tournament could also choose the candidates from the representative set itself, by either of the two methods above. Nevertheless, the implementation here is easier to analyze and should have comparable performance characteristics.

11.3.2 Equivalence Class Sharing

Equivalence class sharing is derived from normal fitness sharing, as it is applied in a single criterion instance where multiple optima of interest exist. There the goal is to distribute the ga population among the optima of the search space, with each optimum receiving a fraction of the population proportional to its relative fitness [10]. Normal sharing causes parental selection to pass over members with niche counts greater than their allowable proportion to encourage reproduction in areas with less population coverage. Niche counts (m_i) estimate the crowding in relative regions of the objective space. Their values are calculated for individual members of the population:

$$m_i = \sum_{j=1}^{P} Sh[d((a,b)] \tag{8}$$

where d(a,b) is the distance between members a and b and Sh(d) is the sharing function. The triangular sharing function is used typically, where $Sh(d) = 1 - d/\sigma_{share}$ for $d \le \sigma_{share}$ and $Sh(d) = 0$ for $d > \sigma_{share}$. Here, σ_{share} is the niche radius,

fixed as some minimum desired separation between population members.

An equilibrium should develop when normal sharing is used such that the shared fitness of all niches is equal:

$$\frac{F_a}{m_a} = \frac{F_b}{m_b} \quad \forall \text{ a, b members' niches} \tag{9}$$

When this equality occurs, every optima has an "effective fitness" F/m equal to every other minima of the search space.

In the multicriteria context, however, fitness does not apply. Rather, we are interested in the resolution of **E**. We would like to emphasize reproduction in sparsely covered areas while ignoring areas more densely resolved. By applying sharing as a tie-breaking measure, we assume that the candidates have equivalent domination characteristics, or that they are in the equivalent "class" of solutions as Figure 11-3 shows:

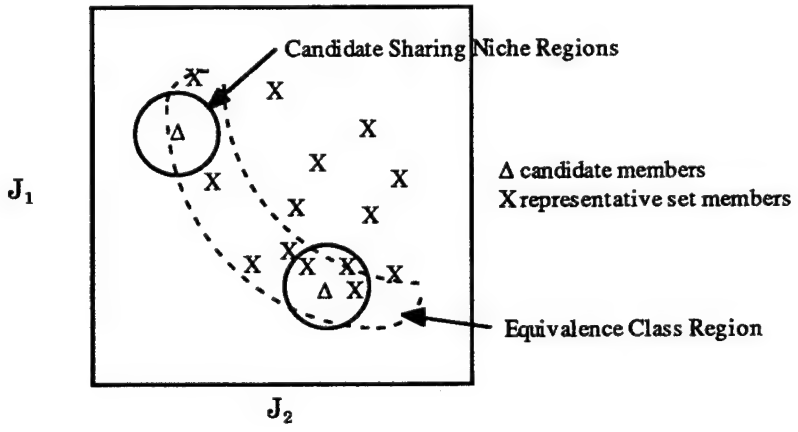


Figure 11-3: Equivalence class sharing

In this multicriteria context where we wish to minimize both criteria, both candidates of the figure dominate the entire representative set. When both candidates dominate the same fraction of the representative set, both lie in the same class of solutions, shown in the figure as the dashed region. To break the tie, equivalence class sharing is used to promote growth in sparsely covered regions of the efficient set. In Figure 11-3, the candidate in the upper left would be chosen as a parent because it has a niche count of zero, which is less than the other's niche count of three.

Equivalence class sharing is performed in the attribute space (attributes are the problem criteria) instead of the genotypic (parameter)

space because the genotype of any problem formulation is very generic in nature and cannot be depended on to provide the distributed representation of the efficient set—which is in the attribute space not the genotypic space.

Criteria Scaling

Multicriteria optimization is performed when attribute scales are non-commensurate (i.e. different units). If some form of scaling of the attributes is not done, the non-commensurate nature of the criteria have a devastating effect on the distribution of individuals in sharing. Sharing in two dimensions, for instance, between a criterion with a range of 10,000 units(a) and one with a range of 1 unit(b) would only spread the population along units(a)!

To avoid unintentional biasing of competing criteria, the attributes are scaled to have non-dimensional units in the range 0 to 1, corresponding to the minimums and maximums of the present population.

Niche size (σ_{share})

The size of the niche regions (circles in Figure 11-3) significantly affect sharing [10]. The MCGA attempts to develop a population evenly distributed along the entire efficient set (**E**). In other words, it seeks to create a discrete representation of the possibly continuous "curve" of non-dominated solutions. Consequently, the appropriate niche size can be thought of as the total "area" of the efficient set divided by the population size (P):

$$\sigma_{share} = E_{area} / P \tag{10}$$

The term "area" is used because **E** is an l-1 dimensional surface in an objective space of dimension l, where l is the number of criteria. Equation (10) can be approximated by:

$$(\sigma_{share})^{n-1} \cong E_{area} / P \tag{11}$$

E_{area} is never known exactly, and rarely even known generally. The bounds on the dimensions of **E** are known, however, as long as we know the bounds on the individual criteria (J_i). By knowing the best (J_i optimal) and worst (J_k corresponding to J_i optimal) on each criterion axis, the minimum area

of the efficient set is the hyperplane passing through the extremes. In two dimensions, for example:

$$\mathrm{Min}(E_{area}) = \sqrt{\left|J_1^{best} - J_1^{worst}\right|^2 + \left|J_2^{best} - J_2^{worst}\right|^2} \tag{12}$$

Note that $[J_1^{best}, J_2^{worst}]$ forms a *single* point in the objective space. The upper bound on the area must be reached asymptotically:

$$\mathrm{Max}(E_{area}) < \left|J_1^{best} - J_1^{worst}\right| + \left|J_2^{best} - J_2^{worst}\right| \tag{13}$$

This is an asymptotic bound because the definition of an efficient set requires that the surface be monotonic (i.e. all first-order partial derivatives have the same sign throughout).

Scaling the attributes from 0 to 1 gives an upper bound of 2/P and a lower bound of $\sqrt{(2)}/P$ for the sharing radius (σ_{share}).

Niche shape

The shape of the niche is affected by the degree of the Holder metric (p):

$$d(a,b) = \left[\sum_{i=1}^{l} \left|J_i^a - J_i^b\right|^p\right]^{\frac{1}{p}} \tag{14}$$

where d(a,b) is the distance between strings a and b in an l criteria objective space.

Figure 11-4 shows the shape of the niche as the degree of p varies.

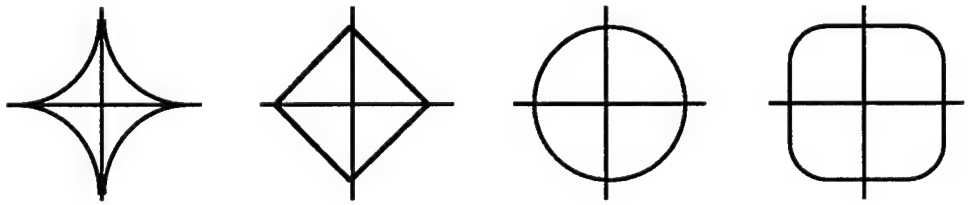


Figure 11-4: Two dimensional Holder metric niche shapes

The limit of the metric at p → infinity is the absolute value of the largest component of the sum above. In Figure 11-4 this would be a square niche.

With Holder metrics of degree $p < 2$, niches along diagonal lines are more densely packed than niches along individual criterion axes. Using this type of niche should provide a higher distribution of individual members along parts of **E** with the smallest criteria tradeoffs. These regions are notable because they signify the least tradeoff between criteria. In problems where most of the set is composed of trading a lot of one attribute for a slight gain in another, a "knee" (a sharp turn of **E**) may result which has an approximate one-to-one tradeoff. Niches with $p < 2$ should place more individuals at "knees" to better explore them [10].

The full MCGA reproduction cycle, as applied in this thesis, is shown in Figure 11-5.

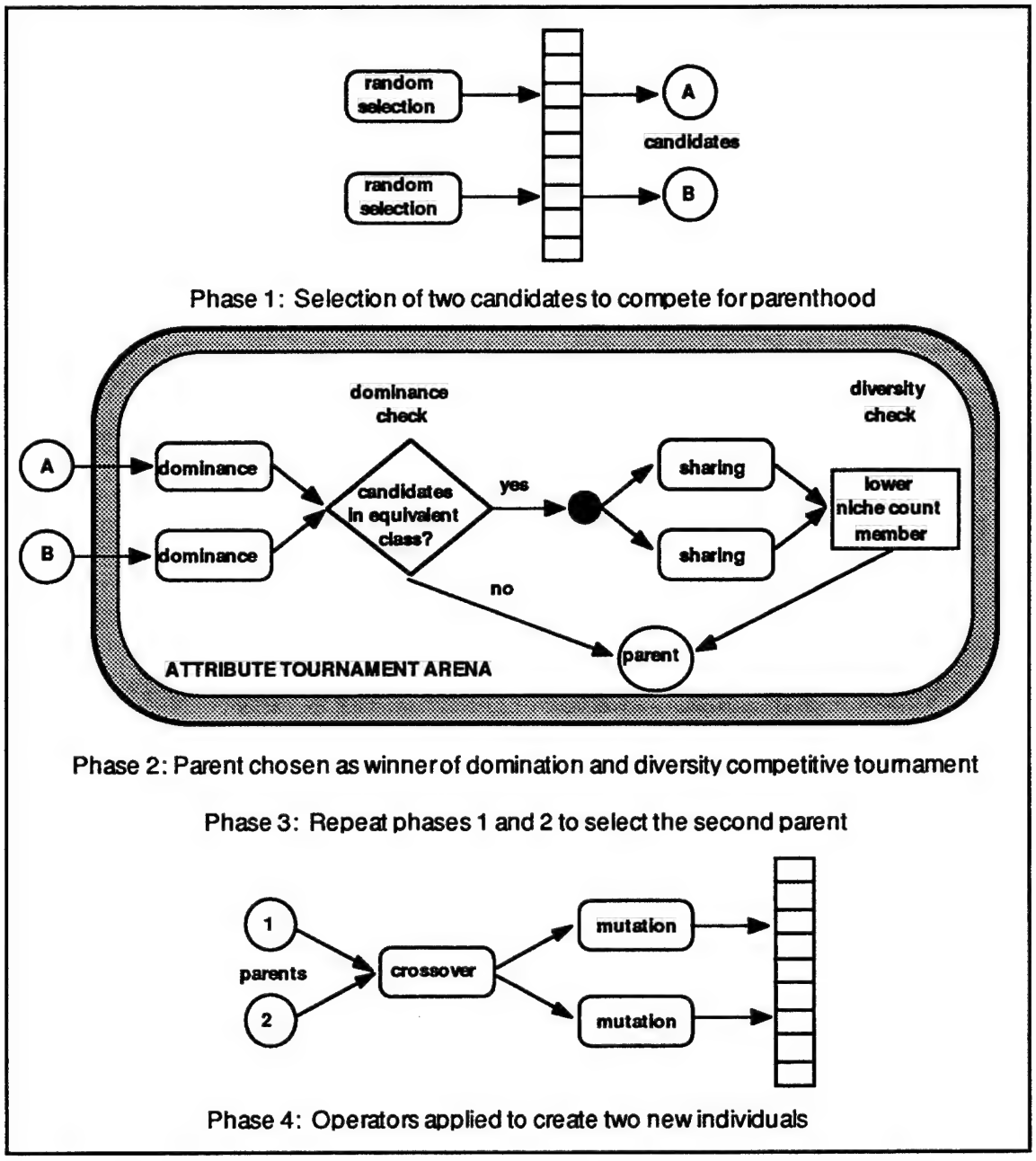


Figure 11-5: Multicriteria genetic algorithm reproduction cycle

Figure 11-5 shows the full MCGA cycle as it produces two children. Phase 2 marks the method's departure from its single criterion cousins and contains the elements of interest for the MCGA parameter analyses of this thesis.

12.0 MCGA Performance Parameters

The implementation of the MCGA, as with any optimization tool, requires the determination of optimal configuration settings necessary for optimal method performance. As has been stated throughout this thesis, we wish to perform optimization using tools with a great deal of *robustness* across fault tolerant system design problems (see Section 3.1 for a definition of robustness). In order to create a robust multicriteria genetic algorithm method, we must first become familiar with the potential configurations that affect MCGA performance.

12.1 Clones

Creating the initial population by random parameter generation injects maximum diversity into the population, allowing the ga to use implicit parallelism to investigate many portions of the design space simultaneously. The robustness of the ga depends heavily on the ability of the reproduction, crossover, and mutation operators to balance diversity (exploration) and efficacy (efficiency). Clones, defined as duplicate strings within a present population, reduce the ga's ability to do both in multicriteria optimization.

In single criterion ga optimization, the ga creates a *single* solution. Since only one solution is desired, the existence of clones in the population may affect the manner in which the solution is attained, but may affect the quality of the solution attained only marginally. Chapter 17 discusses the pros and cons of eliminating clones from single criterion ssga optimization.

In multicriteria optimization, on the other hand, the ga creates a population of points as the problem solution. Assuming a fixed population size, the existence of clones reduces the ga's exploration of the design space by limiting diversity and reduces efficiency by hindering the available resolution point of the efficient set. With a population size fixed to X members, for instance, an MCGA that prevents cloning can theoretically create an X point resolution of the problem's efficient set, while the same MCGA without clone prevention would be bounded to X minus the current number of clones in the population. The number of clones that appear in MCGA operation is explored in Chapter 14, but foresight of their potential ill effects enhances the analysis and development of the MCGA for fault tolerant system design.

Three cases are examined to see the effects of cloning on MCGA performance:

Allow cloning

This control case makes no provisions for the existence of clones in the population. They are not treated any differently than other population members and are allowed to remain and reproduce.

Clone removal

The strictest means of dealing with clones is to remove them directly as they are produced by crossover or mutation. This requires checking new members against the *entire* population as they are created. To implement this case, the two children produced by a reproduction cycle are checked, if either is a clone, both are removed, and the reproduction cycle is repeated until two unique members are created. This case ensures maximum diversity in the population at the expense of many string comparisons and reproduction cycles.

Clone penalization

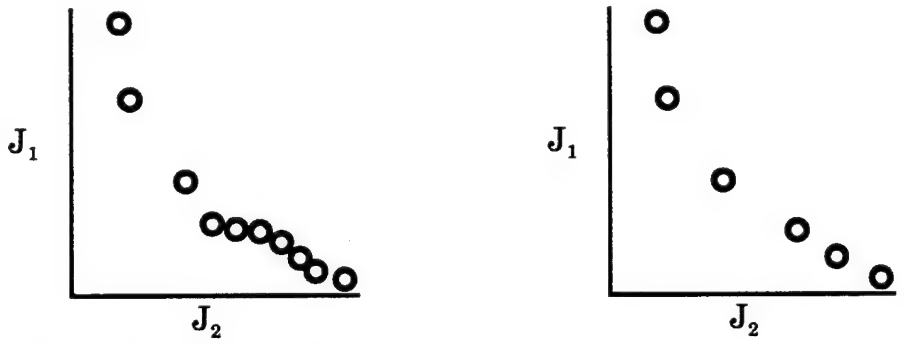
The final case of dealing with clones identifies them at each reproduction cycle by checking new members against the entire population and allows them to be placed in the population. As always, the reproduction cycle selects strings randomly and removes those with low "value" to make room for new strings. Clones are "penalized" because they are removed from the population

if they are selected for "value" testing and they never reproduce because they are prohibited from being selected as parental candidates. Clone penalization reduces the heavy computational cost of repeating the reproduction cycle for each clone created, but the existence of some clones in the population may affect the efficient set resolution.

12.2 Population Variability

As stated earlier, the goal of the MCGA is to generate a high resolution representation (**e**) of the design problem's efficient set (**E**). Since the decision maker (DM) often chooses the route of multicriteria optimization in the first place due to lack of knowledge about tradeoffs between problem criteria, the DM and analyst usually do not know a priori if the proper resolution is 100 points or 10,000 points.

This uncertainty about resolution hinders DM confidence in the constraint and weighting methods. The DM would often prefer the problem resolution produced by generating techniques to provide a good sense of the potential tradeoffs necessary at different parts of the efficient set. Figure 12-1 illustrates an efficient set for a two dimensional discrete parameter problem where generating techniques that divide the efficient set resolution a priori may mislead the DM.



(a) actual 10 point efficient set

(b) constraint method representation of same efficient set (10 point resolution)

Figure 12-1: Example efficient set resolution limitations

A generating method capable of adapting to each problem's unique efficient set would show the concentration of efficient solutions that the constraint method misses in Figure 12-1. Flexibility to adapt **e** to the actual

resolution as it evolves has great advantages. The constraint method in Figure 12-1 is instructed to produce 10 points in the solution, the uneven distribution in the actual efficient set caused 4 cycles of the constraint method to generate duplicate solutions. Though this example is contrived to illustrate a point, it shows how inflexibility in resolution leads to significant waste of computational effort.

The MCGA has been developed to incorporate the possibility of adaptive population dynamics to encourage flexibility in its efficient set resolution. Three cases are examined: fixed population size and two types of variable population size. These cases are covered later, after the process for adding and removing (killing) members of the population is explained.

Adding members

Every reproduction cycle of the MCGA produces two children. The number of members added to the population in each cycle is therefore always two.

Killing members

The process of identifying members of the population that should be deleted each reproduction cycle depends on the cloning parameter in use and the configuration of the tournament (t_{dom}) set. If clone penalization is used, clones identified during the creation of the t_{dom} set are marked for removal and are not included in the tournament set. Clone elimination never allows clones in the population, so no clones would be identified at this point. Additional members to be killed are chosen by identifying strings of the t_{dom} set that are completely dominated by candidate members. This comparison occurs during the tournament arena phase of reproduction (see Figure 11-5). A single tournament set is used to determine the dominance for two parents in a reproduction cycle. Since each parent is chosen as the winner of the tournament between two candidates, each t_{dom} member has four opportunities to be declared a dominated member and marked for removal.

Now that we understand how strings are added and removed from the population, we can explore when these processes are invoked. The actual change in the population size (P) during each reproduction cycle depends on the

number of strings marked for adding and deleting and the P dynamics option chosen, as described in the following paragraphs.

12.2.1 Fixed population size

This is the control case. It requires a single, fixed population size. The appropriate P must not only consider the diversity and computational guidelines that the single criterion ga deals with (see Chapter 8), but must also consider the anticipated or desired **e**. Fixing P requires that exactly two members be killed each cycle by clone and/or dominance checking. If less than two are marked, members must be selected at random from the population or the new members cannot be used. Random selection of killable members is used in this thesis, but neither option is clearly superior since random selection entails the loss of potentially valuable genetic information, while not using the new strings keeps the population from evolving.

12.2.2 Variable population size (±2)

This case allows the MCGA population size to vary from +2 to -2 of its present size at each reproduction cycle. It intends to take advantage of evolutionary pressure to dictate the population size—and the efficient set resolution as a consequence.

The amount of increase is determined as follows:

$$\begin{aligned} &\Delta P = a - b \\ \text{where}\quad &a = \text{added members} = 2 \\ &b = \text{lesser of 4 and killable members} \end{aligned} \tag{1}$$

If no members are marked for removal, P rises by two, while if four or more members are marked, the population size decreases by two. The limitations on ΔP limit the collapse rate of the population to reduce the tendency to eliminate beneficial diversity from the population.

12.2.3 Variable population size (+2/-P_{dom})

This final case allows more drastic fluctuations of the population size by increasing the maximum population collapse rate to P_{dom} members, where P_{dom} is the domination tournament fraction (t_{dom}) multiplied by the full population

size (P) (see Section 11.3.1). The P_{dom} bound on population size decreases allows a higher P collapse rate if the method desires one, while still limiting the loss of diversity during any one reproduction cycle. The P_{dom} value is chosen to represent a known fraction of the overall population size significantly larger than the limitation of -2 of the (±2) case above.

No matter which case of variable population size is used, though, other operations of the MCGA are affected. The size of the niche radius σ_{share}, defined in Section 11.3.2, and the domination tournament set t_{dom} (Section 11.3.1) depend on P. Whether σ_{share} and/or t_{dom} are kept fixed or varied with P may affect MCGA performance. These quantities are allowed to vary with the P in this thesis. This decision assumes that the population size dominates MCGA performance, and that the size of t_{dom} and σ_{share} should be appropriately correlated. We have not, however, tested this assumption and recommend its verification in future research.

12.3 Definition of a Tie

Though dominance is easily seen as an excellent means of pursuing **E**, the definition of dominance can significantly affect the manner in which the algorithm proceeds. As such, MCGA parental string selection is examined using two ways of declaring a domination tie between candidates. (1) If dominance means a candidate dominates the *entire* tournament set, as suggested by [10], the tournament is very selective since candidates will not be completely dominant in most cases, leading to many ties that have to be decided by equivalence class sharing. Sharing must also be done in this case if both candidates dominate the entire t_{dom} set. This option emphasizes resolution of the efficient set over efficiency of moving toward **E**. (2) Dominance can also be defined as the candidate that dominates the *greatest fraction* of the tournament set. In this manner sharing is only required when both candidates dominate the same number of tournament members—emphasizing efficiency over resolution.

To summarize, the definition of a tie is made in two different manners that require sharing to resolve:

1) one (only) candidate does not dominate *entire* t_{dom} set
2) both candidates dominate *same fraction* of the t_{dom} set

12.4 Tournament Size

The size of the domination tournament set (t_{dom}) is approached in this thesis as a fraction of the full population. Reference [10] found t_{dom} of about 10% of P to be generally adequate for on-line performance (see Section 11.3.1). The impact of this parameter is not investigated by this research. All subsequent MCGA analysis for this thesis uses a t_{dom} of 0.20, or 20 percent of the full population size.

12.5 Niches

The appropriate size and shape of equivalence class sharing niches is not investigated by this thesis. The size of a niche (σ_{share}) is described in Section 11.3.2. This thesis uses

$$\sigma_{share} = \frac{1}{2}\left[\frac{2+\sqrt{2}}{P}\right] \tag{2}$$

which is the value half-way between the upper and lower bounds on the parameter. The shape of the niche depends on the degree p of the Holder Metric used and is also described in Section 11.3.2. This thesis uses p = 1 to place the greatest emphasis on equal tradeoffs between criteria. Also, p = 1 allows the metric to be calculated by a simple sum of absolute values instead of powers and roots, which keeps the computational effort low.

12.6 Summary of Options

Table 12-1 summarizes the various MCGA implementation parameters of interest in this thesis, and the different aspects of performance they influence.

name	definition	values or options	influence
cloning	duplicate strings	allow cloning	reduces efficiency and exploration
		clone removal	high computational effort required
		clone penalization	some clones remain in population
P dynamics	variable population size	fixed P	inflexible efficient set resolution
		variable P (±2)	gentle variability
		variable P (+2/-P_{dom})	strong variability
ties	definition of a tie	one candidate does not dominate entire t_{dom}	emphasizes diversity
		both dominate same fraction of t_{dom}	emphasizes domination
t_{dom}	fraction of P for tournament set	0.2 **[not examined]**	determines domination pressure
sharing niches	sharing radius (σ_{share}); niche shape (p)	$\sigma_{share} = \frac{1}{2}\left[\frac{2+\sqrt{2}}{P}\right]$ p=1 **[not examined]**	medium sized niches that emphasize small criteria tradeoffs

Table 12-1: MCGA implementation parameters

The effect these parameters will have on MCGA performance is examined in the following sections.

13.0 MCGA Performance Criteria

Judging multicriteria method performance is a great deal more difficult than similar efforts with single criterion optimization methods. Objective measures of performance are not clearly defined in any of the references cited on multicriteria optimization [2] [4] [10]. However, the need for concrete measures for determining improved performance when comparing MCGA implementation parameters and for making comparisons to other generating techniques has led to the choice of a number of performance measures. The first four represent concrete measures of multicriteria method quality. The remaining two differentiate performance variations of different MCGA configurations, though they are not be deemed crucial indicators of performance in actual fault tolerant system design optimization.

13.1 Efficient points

The goal of a generating method is to produce an accurate representation of the problem's efficient set (**E**). The resolution of the representation (**e**) must be sufficiently fine to provide the DM with an adequate understanding of the tradeoffs between criteria. As such, the number of efficient points in a method's solution is an upper bound on the number of quality points that the method can have in its solution. If a method produces points that are dominated by members of its own **e**, they are by definition excluded from inclusion in the problem's **E**.

This performance measure is simply the number of points in the method's final solution that are efficient with respect to the remainder of the

solution. Unfortunately, the actual number of points in and the location of **E** in each of our discrete fault tolerant system design test problems are unknown. As such, the number of efficient points in any method's solution can only provide a relative comparison measure of performance. In addition, the fact that **E** is unknown prevents us from making any definitive statements about the position quality of a *method's* efficient set (**e**).

13.2 Efficient set spacing

This measure refers to the variance of the range (distance) of each member of the *current* **e** to its closest neighbor (also of **e**). It is measured in the criteria (phenotypic) space by a p = 1 degree Holder metric (see Figure 11-4) to correspond with the niche measurements. Note that only those strings that lie in the current population's representation of the efficient set (**e**) are included in this measure.

A generating method that minimizes this quantity has good efficiency in finding **E**. The formula of this measure is given in equation (1).

$$f_{spacing} = s^2 = \frac{1}{e-1}\sum_{i=1}^{e}\left(\bar{d} - d_i\right)^2 \qquad (1)$$

$$\text{where} \quad d_i = \min_j\left\{\left|J_1^i - J_1^j\right| + \left|J_2^i - J_2^j\right|\right\}$$

A value of zero for this metric would mean that all members of **e** are equally spaced from one another.

13.3 Seven point distance measure

The MCGA's ability to completely resolve **E** can be quite accurately seen by how close it comes to the individual criteria optima and other known (predetermined) points of **E** if prior knowledge of the efficient set is available. Since **E** is not known for any of the test problem, seven points of comparison are generated for each problem to create a measure of the algorithm's efficacy. The individual criterion optima, which bound the efficient set of the two criteria problem, were found by optimizing each criterion separately without regard for the other. With the resulting two points at hand, the seven comparison points are defined on a J_1-J_2 as the origin [0,0], the maximum (within the range of **E**)

of each criterion [0, J_2^{worst}] and [J_1^{worst}, 0], and two points on each axis between the origin and the maximum value. These points are shown in Figure 13-1.

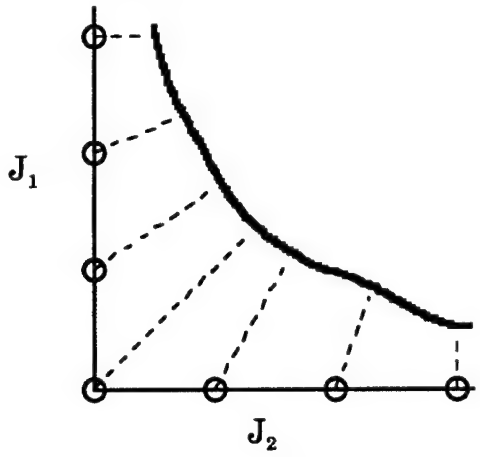


Figure 13-1: Seven point distance measure of population accuracy

The full distance measure is created by averaging the Euclidean distances from each of the seven axis points to the member of the current MCGA population that is closest to each point. Therefore, seven members of the population are used each time the distance measure is created. This figure of merit is an accurate means of comparing the relative dominance of different populations to one another on a particular problem. The population with the smallest distance measure value for a given problem will be the one that most closely approaches **E**.

13.4 Cost Function Evaluations

As with single criterion optimization, the DM is limited by time and computing constraints. Therefore, optimal performance of the MCGA must account for the number of cost function evaluations (cfe) required. The number must be competitive with other multicriteria methods for the MCGA to be worth using. Since a means of gauging convergence to the efficient set is not presently available for any method, the solution quality (number of efficient points, distance measure, spacing) as a function of computational effort will be the main comparison measure of the multicriteria analyses in Chapters 14 and 15.

13.5 Additional Criteria

13.5.1 Proportion of Clones

The proportion of clones in the population is a parameter (genotypic) quantity we wish to minimize, though it is not necessarily a hindrance to the MCGA's ability to create high quality solutions. A value of zero for this measure indicates that no member of the population is a clone of any other member of the current population. Intuitively clones are a hindrance to both efficiency and exploration and should be eliminated, but performing the operations necessary to keep clones from forming may be unnecessarily prohibitive. A configuration with good efficiency, good exploration, and a low fraction of clones is preferable if we do not have to perform excessive operations against the clones that form in the reproduction cycle.

13.5.2 Total clones identified

Depending on how the MCGA deals with clones in the population, a great deal of additional computational effort may be required to keep the desired optimization performance. When comparing different configurations of the MCGA, those that identify the greatest number of duplicate strings in the course of their runs are likely placing the highest amount of effort into the reproductive process.

14.0 MCGA Performance Analysis

The MCGA is an unproved capability at the time of this writing. The ability of the ga in general to perform multicriteria optimization has only had limited analysis. Unlike the single criterion application of the ga, where the basic framework and performance of the ga has been worked out and debated for several years, multicriteria ga implementations are still in their infancy. Therefore, this thesis not only investigates whether ga's can do multicriteria optimization of fault tolerant system design better than other methods, it also investigates the basic viability of ga's in multicriteria optimization.

The majority of the multicriteria analyses shown are performed in two dimensions. The reader must note that unlike comparative methods that suffer dimensionality difficulties, the MCGA suffers no (theoretical) implementation hindrances—the choice of two dimensional analysis is made solely for method comparison and visualization purposes.

The two criteria of interest are the unavailability (i.e. 1.0 - reliability) and the system purchase cost (in dollars). The unavailability is treated as a logarithmic quantity to accentuate the differences at the better (smaller value) edge of performance. The purchase cost, on the other hand, is simply treated as a linear quantity to be minimized.

The proper configuration of the MCGA for effective multicriteria optimization is still very much uncertain. Therefore, the first priority of this thesis is to prove that using a ga with domination as the string selection criterion can actually perform multicriteria optimization. On those lines, Figure 14-1 shows an MCGA attempt to create an efficient set representation

of the TISS problem. The configuration for this problem includes a t_{dom} of 0.2, variable P by (±2) bounds per reproduction cycle, and clone penalization. The MCGA is given an initial P of 1,000, shown in the two dimensional plot of Figure 14-1 by the "+" symbols. After 10,000 cost function evaluations, the population has 349 members and is shown as "o".

This type of plot is used throughout the rest of this analysis. Note that both axes are normalized from 0 to 100+. The normalization values are obtained by optimizing each criterion separately using the steady-state genetic algorithm (ssga). The assumed minimum purchase cost is represented as 0; where unavailability is assigned a value of 100 (i.e. the J_1-J_2 point of [100,0]). The same scaling is used for the other axis based on the minimum of unavailability. Criteria values greater than 100 can occur in dominated solutions of the design space, as seen on the unavailability axis of Figure 14-1. The vertical axis is a normalization of the log of the unavailability values.

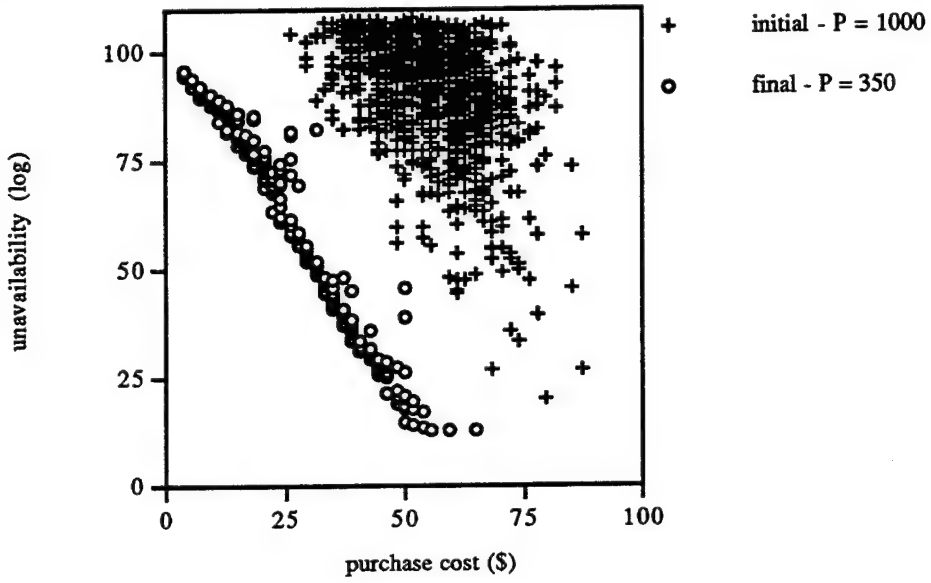


Figure 14-1: Sample MCGA optimization of TISS problem

Obviously, the MCGA in this example at least, moves the population towards the efficient set (**E**) of the problem and improves the quality of its solutions over time. The initial population, though randomly generated to cover a wide area of the parameter space, does not have any points in the vicinity of the final **e**. The MCGA makes a vast improvement over the 1,000 point Monte Carlo attempt (i.e. the initial population creation). The final population appears to have a good spread of members across its whole representation,

but there are no values near the minimum of unavailability, and the purchase cost minimum found by the single criterion ssga is not attained. Note that **E** is unknown and Figure 14-1 is just one possible representation.

Now we must analyze the influence MCGA parameters have on performance and how those parameters can be best set for effective fault tolerant system design. Once that is accomplished, the MCGA can be compared to the constraint method to determine its viability as a competitive multicriteria optimization tool.

14.1 Effect of Clones

The first configuration question to be addressed is the impact of clones on MCGA performance. Implementations of the MCGA on the TRIPLEX and TISS problems is analyzed.

Cloning analysis (Triplex)

The MCGA is run on the TRIPLEX problem for the three clone handling options of allowing clones, clone replacement, and clone penalization. The control configuration for this analysis includes a fixed P of 200 members, t_{dom} of 0.2, and declaring candidate domination ties when only one does not dominate the entire t_{dom} set. Each option is run for 50,000 cfe with 20 different starting points (random number generator seeds).

Some of the effects of each clone option can be seen in Figure 14-2. It shows the fraction of the current MCGA population that is a duplication of other members. Allowing clones to occur in the population has a dramatic effect on the effective proportion of MCGA population. In Figure 14-2, for instance, allowing clones in the population causes upwards of 80% clones to occur. The dramatic rise in the clones occurs in the first 2,000 cfe, but is not captured by the scale of Figure 14-2 (Figure 14-7 illustrates the clone fraction behavior of the early stages of a run). A P with a percentage of clones this high has to be approximately four times larger than a P containing no clones to get the same potential **e**. Clone removal and clone penalization both keep the proportion of clones small: clone removal at 0% and clone penalization never exceeds 5%.

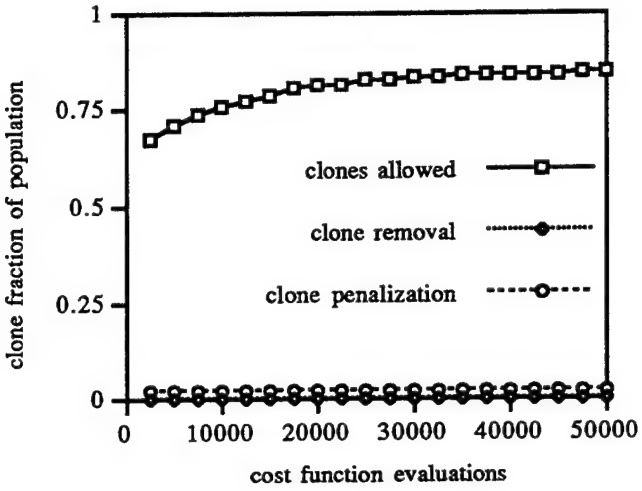


Figure 14-2: Clones in population for 3 clone options on TRIPLEX problem

Now that a definite impact of changing the clone option has been identified, two issues remain. The first is the amount of computational effort expended for clone identification and marking, while the second, and more important, is the performance impact each clone option has. Figure 14-3 illustrates the first issue by showing the number of clones marked by each option over the MCGA run.

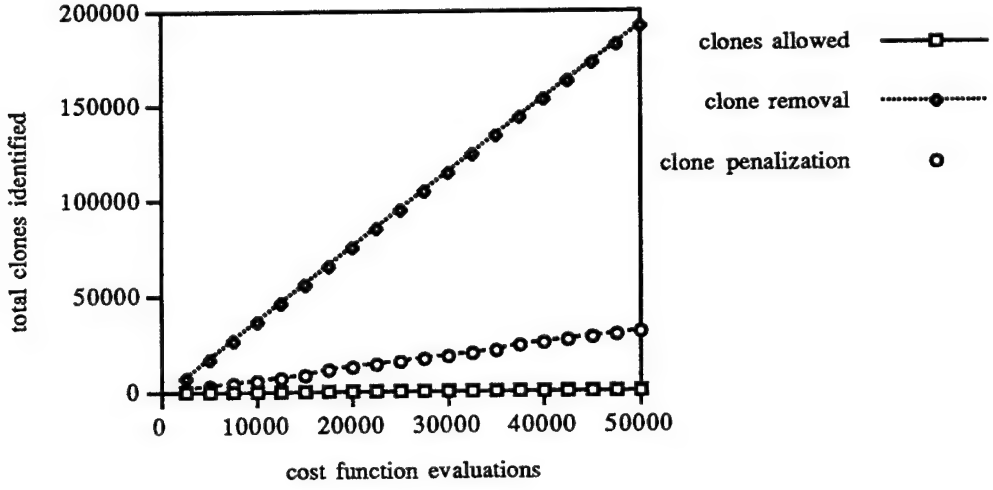


Figure 14-3: Clones identified in TRIPLEX problem for 3 clone options

The "clones allowed" option does not check for clones and obviously shows zero effort required for dealing with clones. The "clone penalization" option shows that approximately 40,000 clones are identified and penalized over the course of the 50,000 cfe TRIPLEX run. As was shown in Figure 14-2,

though, strict penalization keeps the number of clones in the population small. The greatest amount of computational effort comes from the "clone removal" option. If 190,000 clones are detected over the course of the TRIPLEX runs, 190,000 extra MCGA reproduction cycles are performed to replace those clones. Though the additional reproduction cycles do not include cost function evaluations, which are the greatest item of time for the method, the high numbers of extra cycles add enough effort to make this option viable only if it exhibits significantly better performance than clone penalization.

The last issue, and most critical, is performance. Figure 14-4 shows the number of points that are efficient to the rest of the population. Note that the quality of the efficient points is not shown by this figure—solution quality is addressed in Figure 14-6.

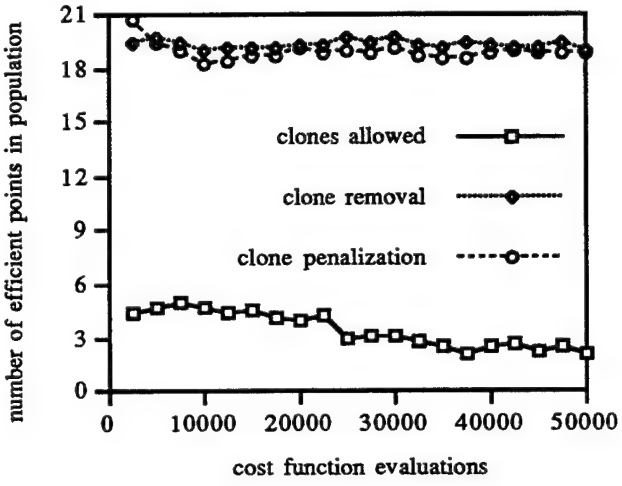


Figure 14-4: Efficient point analysis for cloning on TRIPLEX problem

As could be expected from Figure 14-2, allowing clones significantly reduces the effective P and keeps the number of efficient points in the population much lower than the other options. Clone removal and clone penalization exhibit very similar efficient set sizes throughout the 50,000 cfe attempt on the TRIPLEX problem. Both options suggest that this simple two criteria design problem has an **E** containing approximately 20 points.

Another measure of performance is how evenly spread **e** becomes. The variance of the range of each member of **e** to the closest member of **e** is shown in Figure 14-5.

This figure shows that allowing clones creates a very poorly spread efficient set representation. This effect is obviously caused by the low efficient

set size seen in Figure 14-4. The other two options appear to have equally well spaced **e**. This figure shows that the solution points created by each method are spread across an efficient front and not clustered in particular areas.

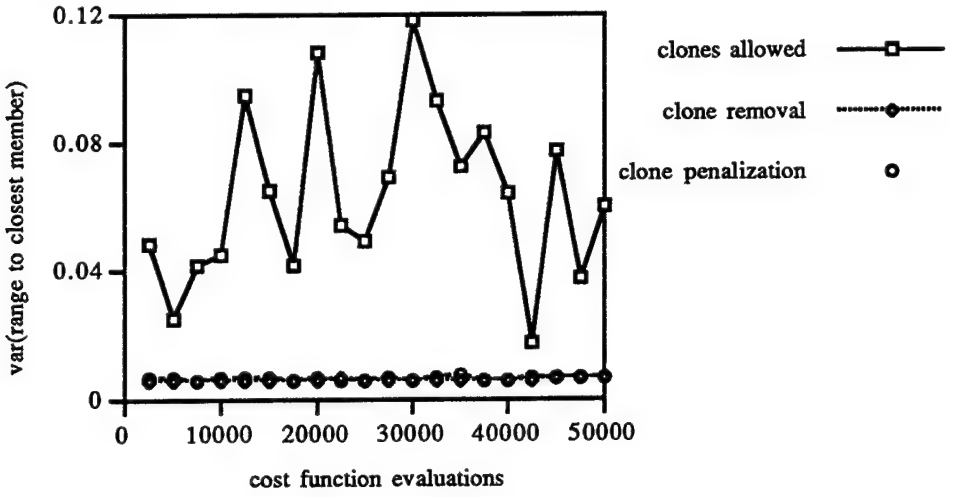


Figure 14-5: Efficient set range variance for cloning on TRIPLEX problem

The final performance measure is the distance from seven points on the criteria axes (see Figure 13-1) to the closest members of the population and is shown in Figure 14-6.

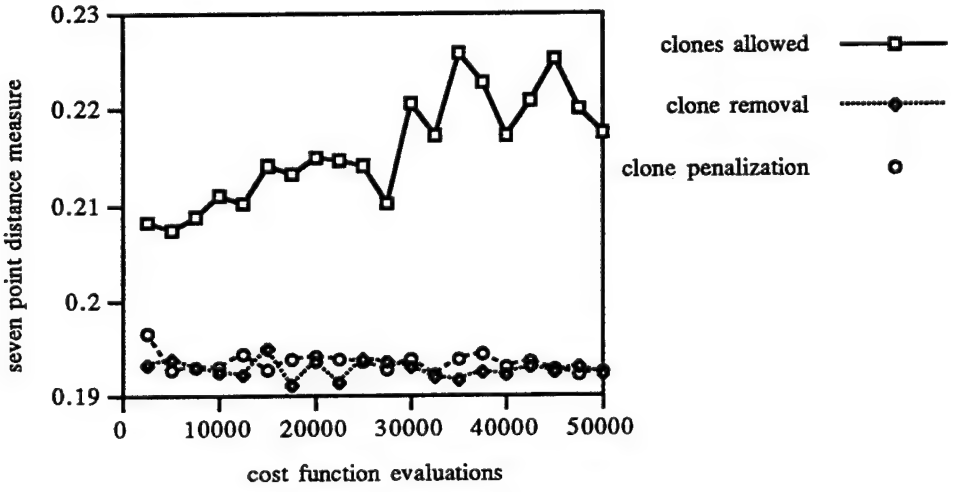


Figure 14-6: Distance measure analysis for cloning on TRIPLEX problem

Because **E** for this problem is unknown, the only the relative distances shown are significant. The scale of Figure 14-6 has what appears to be a narrow range of normalized average distances between 0.19 and 0.23. Absolute determination notwithstanding, however, the clone removal and clone

penalization options again exhibit similar performance that is superior to that of allowing clones.

The performance of clone removal and clone penalization matches very closely in all of the performance plots for the TRIPLEX problem, but the additional computational effort required by clone removal places it at a disadvantage.

Cloning analysis (TISS)

A single set of data from one, very simple fault tolerant system design problem is not necessarily generalizable. As such, the results of the same clone handling options are now shown for the TISS problem before any conclusions are drawn. The control configuration for the TISS analysis includes a fixed P of 600 members, t_{dom} of 0.2, and declaring candidate domination ties when only one does not dominate the entire t_{dom} set. Each option is run for 20,000 cfe with 20 different random number generator seeds.

Some of the effects of each clone option can be seen in Figure 14-7 and Figure 14-8. They show the fraction of the current MCGA population that is a duplication of other members and the total number of clones detected.

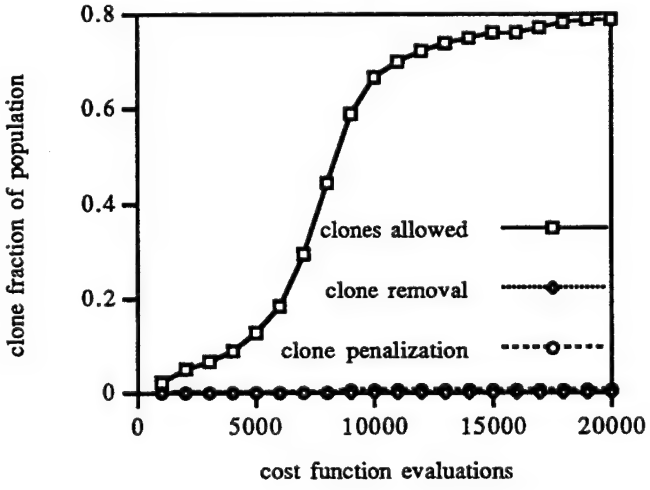


Figure 14-7: Clones in population for 3 clone options on TISS problem

This figure shows that allowing clones to appear unimpeded significantly affects the effective population size.

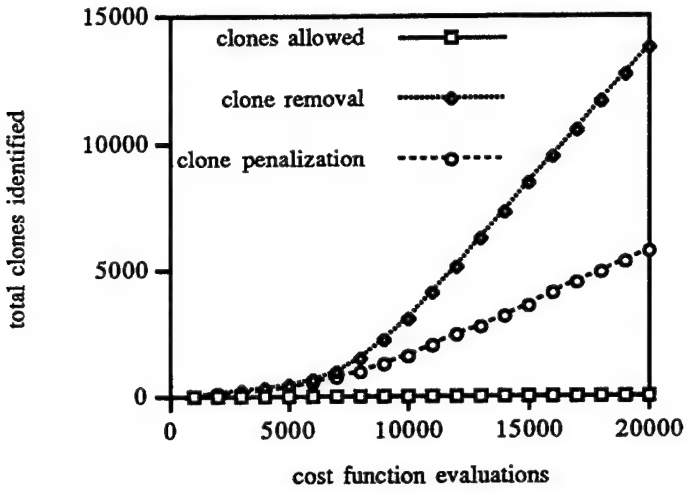


Figure 14-8: Clones identified in TISS problem for 3 clone options

The additional problem complexity in the TISS problem over the TRIPLEX problem is seen by the far fewer clones detected in Figure 14-8 as compared to the high numbers shown in Figure 14-3. As is evident for the previous example, Figure 14-8 also shows that removing clones from the population as they occur forces a significant number of additional reproduction cycles to be performed even though it does not keep the number of clones in the population much lower than clone penalization.

The effects of the three clone handling options on the TISS problem are shown in the following three performance plots.

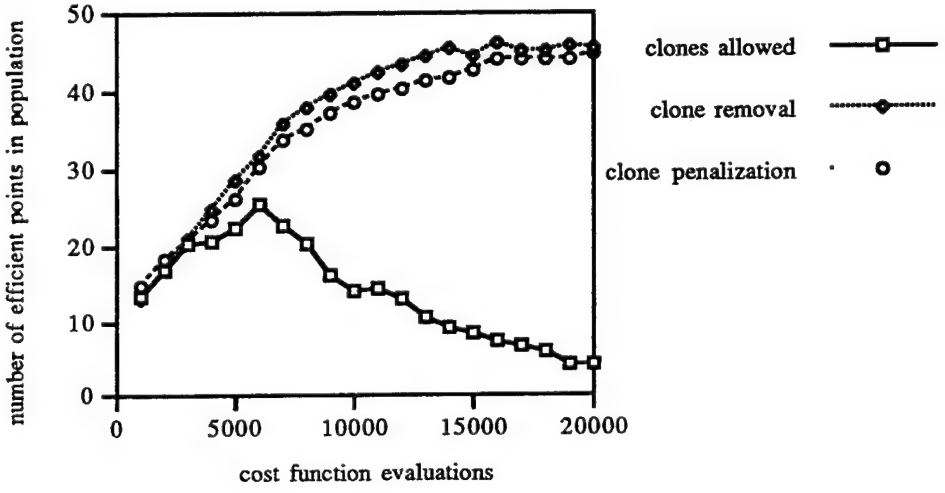


Figure 14-9: Efficient point comparison of cloning on TISS problem

Figure 14-9 shows that allowing clones to crowd out unique members of the population reduces the number points in **e**. The other two options improve the number of efficient points, with clone penalization having a smaller **e** for a given number of cfe, but not significantly so.

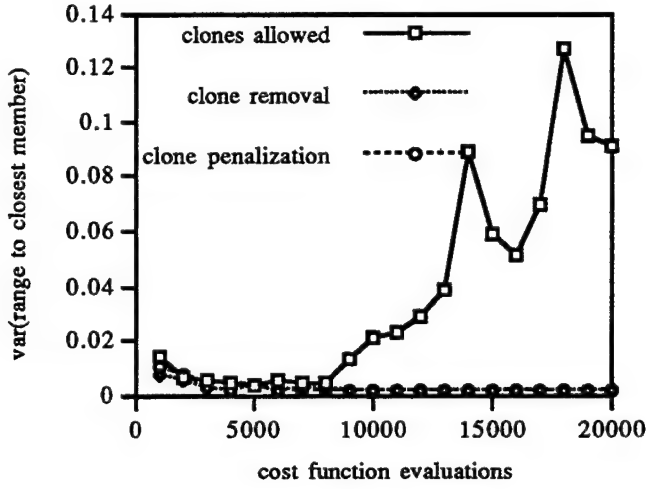


Figure 14-10: Efficient set range variance of cloning on TISS problem

Figure 14-10 shows that the spacing of **e** for clone removal and clone penalization becomes better with time, while allowing clones exhibits poor performance after about 7,500 cfe.

The final performance measure comparison necessary to determine the appropriate clone option is the distance measure for the TISS problem.

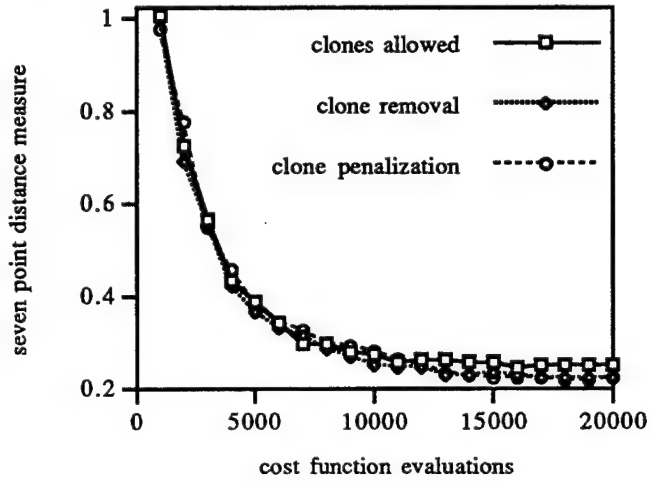


Figure 14-11: Distance measure analysis of cloning on TISS problem

Figure 14-11 shows an interesting, and somewhat unexpected result. Though it has displayed significantly worse performance to this point of the analysis, "allowing clones" on the TISS problem shows the same general characteristics with respect to its ability to represent key portions of the efficient set. Though the others eventually show better distance measures at the end of the run, the difference is not apparent until 10,000 cfe have been performed.

The scaling of Figure 14-11 is somewhat misleading because of the poor performance all options have with their initial populations. The difference between the options is more significant on the TISS problem than the figure would indicate. Figure 14-12 shows the distance measure again using a condensed scaling to accentuate the differences between the options during the end of the runs.

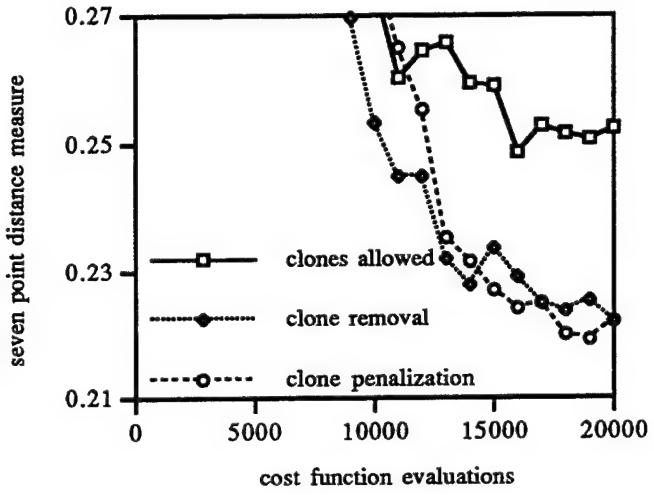


Figure 14-12: Distance measure for cloning on TISS problem (end of run)

The differences between options in Figure 14-12 indicate that even though Figure 14-11 shows the same general behavior for all three options, the differences between them at the end of the run are significant. Again, as stated earlier, the distance measure only shows relative performance, but again the "allowing clones" option is outperformed by the others. In conclusion, therefore, the poor overall performance of the "allowing clones" option in both the TRIPLEX and TISS problems argues against its use.

As in the TRIPLEX analysis, the solution quality does not appear to be significantly different between the "clone removal" and "clone penalization" options. Allowing identified clones to remain in the population but prevented

from mating and encouraged to be replaced by new members appears to have the same performance effect as removing clones outright. The difference between the two approaches comes in the amount of computational effort required to achieve their relative goals. Clone removal requires four times as many reproduction cycles in both the TRIPLEX and TISS problems as clone penalization. As such, clone penalization is used in the control configuration for the remainder of the MCGA analyses of this thesis.

14.2 Population Variability

As discussed in Section 12.2, multicriteria optimization is often performed in a fault tolerant system design due to the uncertainties the DM has about the relationships between criteria. As such, the proper resolution of a particular **E** is not known prior to optimization, forcing the DM or analyst to make arbitrary assumptions about the problem when resolution is an input to the algorithm, such as it is in both the constraint and weighting methods. The MCGA retains considerable latitude in its resolution of the efficient set. For instance, a P of 100 members can create 5 point or 100 point **e** if a particular problem warrants one or the other.

This section of the MCGA analysis, however, takes the efficient set resolution (**e**) one step further by allowing the MCGA to vary its population size based on preset conditions detailed in Section 12.2 to determine if the MCGA can optimize its resolution on each problem it encounters.

Three population variability options are analyzed using the TRIPLEX and TISS problems. Both problems are optimized on unavailability (logarithmically scaled) and purchase cost. The MCGA's performance on the TRIPLEX problem is analyzed first, followed by the TISS problem, and the applicability of population variability is discussed at the end of this section.

P variability analysis (Triplex)

The MCGA is run on the TRIPLEX problem for the three population variability options of fixed population size (P), variable P with (±2) bounds, and variable P with (+2/-P_{dom}) bounds. The control configuration for this analysis includes an initial P of 200 members, t_{dom} of 0.2, declaring candidate domination ties when only one does not dominate the entire t_{dom} set, and clone penalization.

Each option is run for 50,000 cfe with 20 different random number generator seeds.

Some of the effects of automatic population variability are visible in Figure 14-13. It shows the current MCGA population size as a function of the cost function evaluations performed in the run.

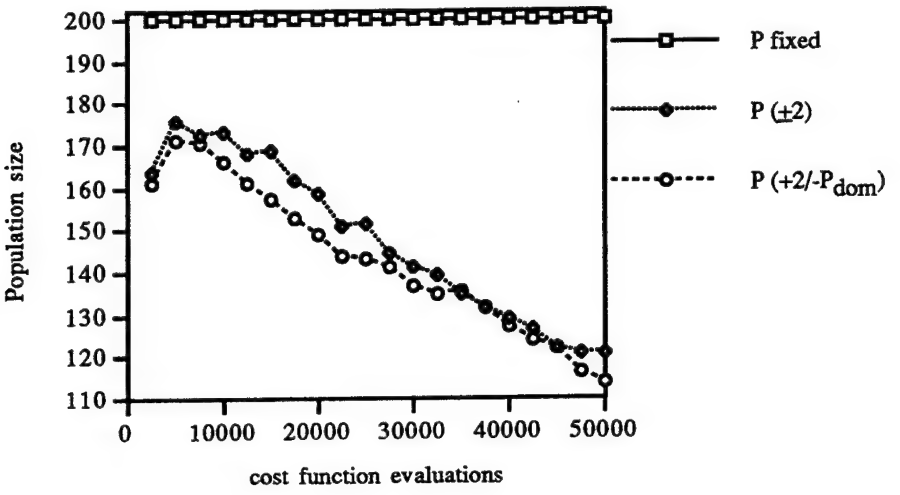


Figure 14-13: Population size with P variability on TRIPLEX problem

This figure shows that allowing population size variability significantly impacts the P retained by the MCGA. The two variable P option reduce their population sizes significantly in the initial portion of the run—prior to the first point shown in the figure at 2,000 cfe. In both cases, P reduces from the initial 200 members to about 160 members. In the next 5,000 cfe of the run, the populations increase in size at similar rates. Following that short increase, both variable populations decrease for the remainder of the run, ending with 120 to 125 members. Though both are similar, the (+2/-P_{dom}) option maintains a slightly lower P, which would be expected since it is allowed to drop members faster than the (±2) option, but cannot add them any faster.

Figure 14-13 shows some general behavior of the variable population sizes, but it says nothing about performance. The following figures show the performance impact of the rising and dropping population size.

The behavior indicated in Figure 14-13 and Figure 14-14 appear to be inversely related with respect to the two variable P options. The initial decrease in the MCGA population size results in **e** being three members smaller on average than for the fixed P option (Figure 14-13). Likewise, as the

runs progress and the population sizes decrease, the number of efficient members increase to be comparable to the fixed P (Figure 14-14).

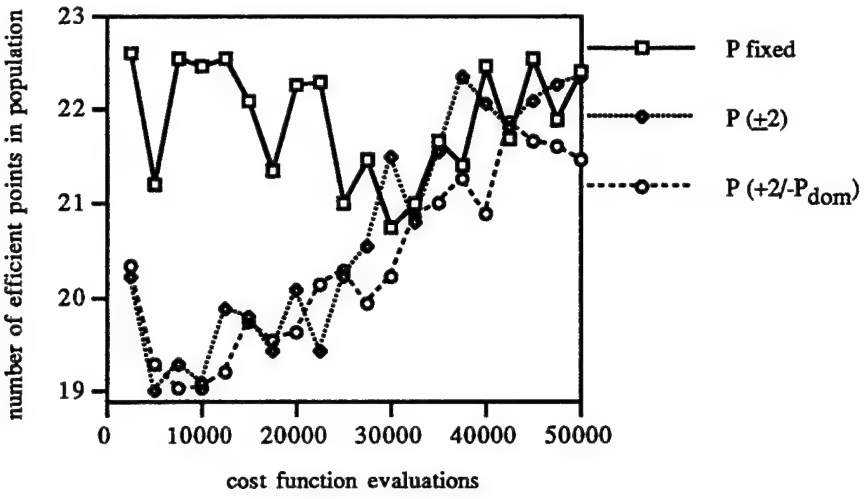


Figure 14-14: Size of **e** with P variability on TRIPLEX problem

The fixed P option keeps an efficient set between 21 and 23 members throughout, but the considerable fluctuations over 20 seeds, may indicate some performance uncertainty in individual runs.

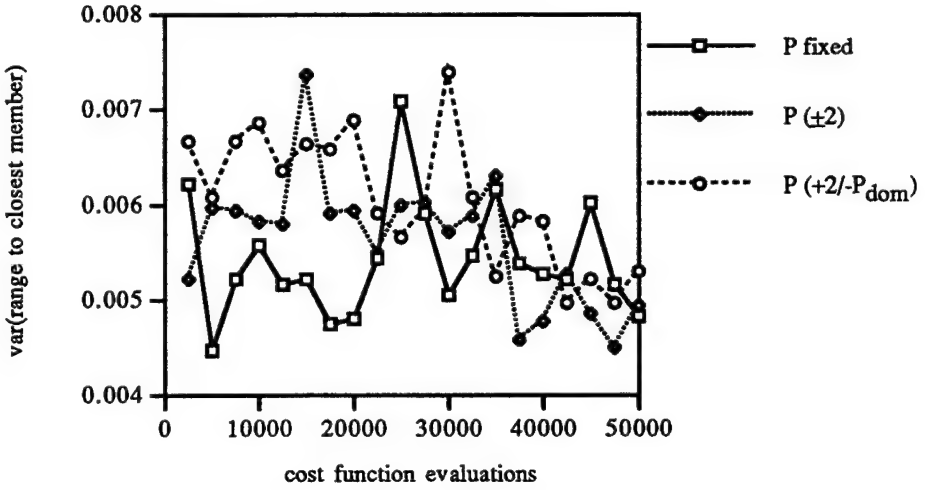


Figure 14-15: **e** range variance for P variability on TRIPLEX problem

Figure 14-15 shows that no significant impact can be detected on the spacing of **e** when P variability is included. All three options are characterized by wide fluctuations within a small variance range that is comparable to the TRIPLEX variance values of Figure 14-5.

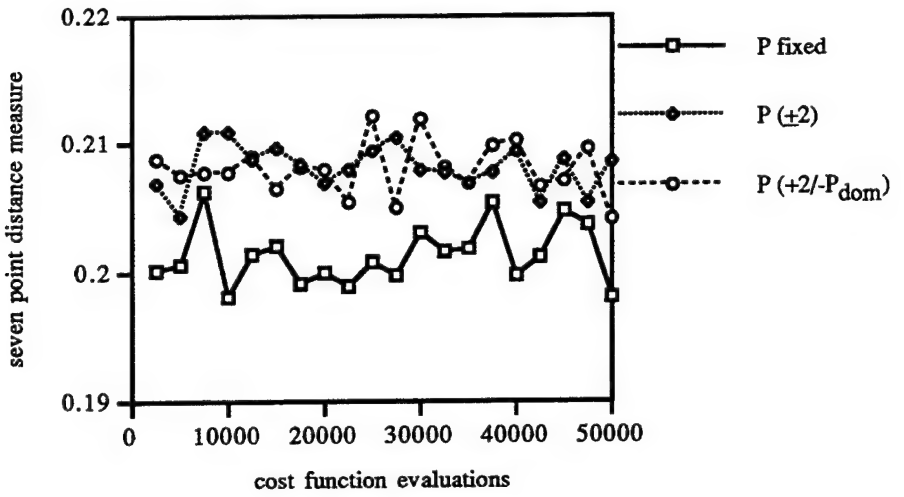


Figure 14-16: Distance measure of P variability on TRIPLEX problem

Figure 14-16 is the final MCGA performance figure for population variability on the TRIPLEX problem. It shows the most significant differences between the three options. The fixed population size of 200 members maintains its distance measure below those of the variable P options throughout the run. There appears to be no difference between the accuracy (distance measure) performance of the two variable P options.

P variability analysis (TISS)

In order to draw broader conclusions on the relative performance of population size variability in the MCGA, the same three P variability options addressed for the TRIPLEX problem are again compared below for the TISS problem. The control configuration for this analysis includes an initial P of 600 members, t_{dom} of 0.2, declaring candidate domination ties when only one does not dominated the entire t_{dom} set, and clone penalization. Each option is run for 10,000 cfe with 20 different random number generator seeds.

Figure 14-17 shows the effect of P variability on the value of P over the course of a run on the TISS problem. As is the case on the TRIPLEX problem, the variable P options initially decrease P dramatically. In the first 500 cost function evaluations beyond the initial creation of the 600 member population, both variable P options dropped their populations to about 100 members, a decrease of about 500 members! From that point to the end of the run, those

two options exhibit parallel performance. Both slowly increase their population sizes to about 200 members by the end of the run.

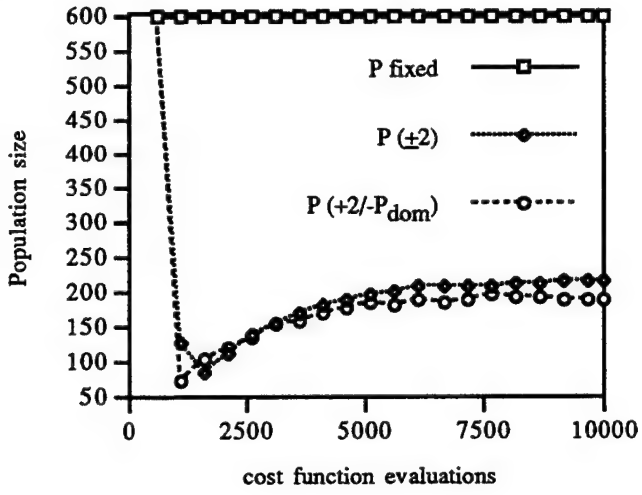


Figure 14-17: Population size with P variability on TISS problem

Using Figure 14-17 as a guide, the MCGA apparently "believes" that sufficient population diversity for the TISS problem can be maintained in a population of about 200 members. If its performance supports that assertion, the MCGA can be used without concerning the user with choosing the optimum initial population size for high performance.

Figure 14-18 shows the number of points **e**.

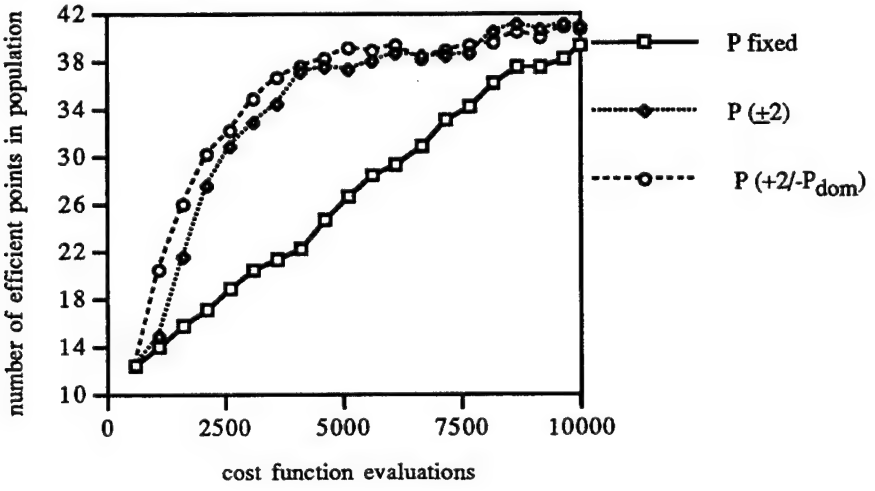


Figure 14-18: Size of **e** comparison for P variability on TISS problem

The results shown in this figure are rather dramatic. All three options begin with approximately 12 efficient members in their population, and all

three end with about 40 after 10,000 cfe. However, the fixed P option increases the number of efficient members at a steady, linear rate, in marked contrast with the variable P options. Even though the population sizes for the variable P options drop dramatically during the initial portion of the run, the number of efficient points in the population rises to about 40 members in only about 5,000 cfe. Of course, the figure above says nothing about the quality of those points, merely that they are efficient. A measure of the distance that **e** is from **E** is illustrated in Figure 14-20.

Figure 14-19 shows the spacing of **e** for the three P variability options.

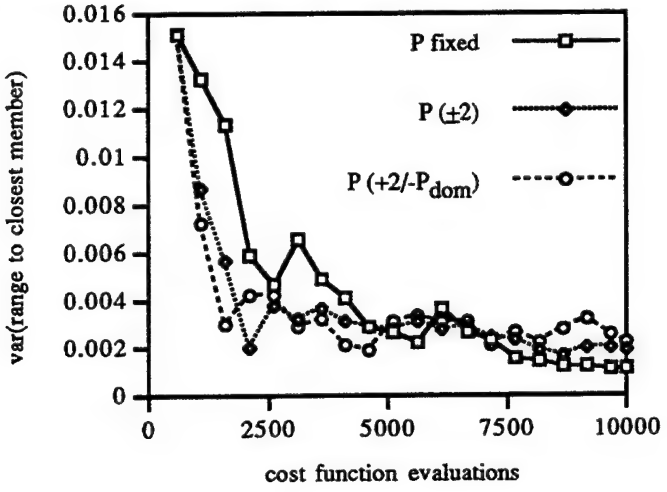


Figure 14-19: **e** range variance for P variability on TISS problem

All three options exhibit similar behavior. The variable P options appear to have better spacing during the early stages of the run, but since the fixed P one has a smaller efficient set, this fact is understandable. The variance of the spacing in the latter portions of the run appears to slightly favor the fixed P option, but the small scaling of values in the figure cautions against making early conclusions about relative superiority.

The final figure for TISS problem performance with P variability shows the distance measure characteristics that indicate solutions quality. As is the case with the TRIPLEX problem analysis for P variability, the distance measure figure of merit proves to be significant in the comparison of the three P variability options on the TISS problem. Though the variable P options holds more points in **e** than the fixed P option for most of the 10,000 cfe run, the quality of those points is better in the fixed P option for the entire run.

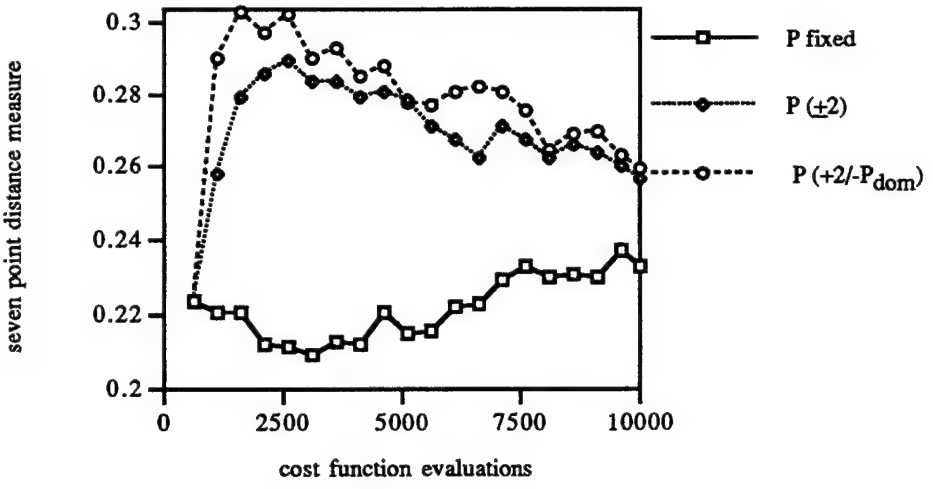


Figure 14-20: Distance measure analysis for variable P on TISS problem

The quality of the variable P options appears to suffer most during the initial stages of the run when the population size fluctuates the most. The longer the run progresses, however, the better both of them become. The general form of the figure suggests that the fixed and variable P lines are converging upon one another and are in the same general range of accuracy at 10,000 cfe.

Finally, there is the question of what effect, if any, P variability has on the number of clones generated by the MCGA. This effect is shown in Figure 14-21 for the TISS problem.

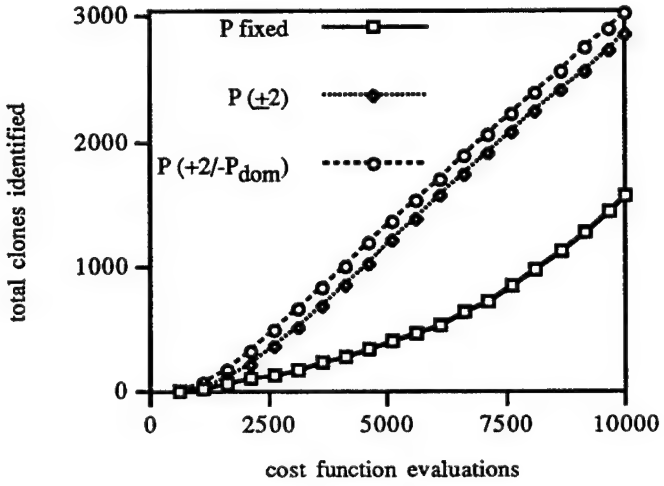


Figure 14-21: Clones detected in TISS problem with variable P

This figure shows that in 10,000 cfe of the TISS problem, varying P causes the MCGA to encounter twice as many clones. Using clone penalization, no computational effort is expended on those clones beyond their identification, and the percentage of clones in the population remains small throughout the run.

The overall results of the TRIPLEX and TISS problem analyses leads us to conclude that a properly chosen fixed population size is generally superior to allowing P to vary. The general effect shown in Figure 14-18 and Figure 14-20, however, indicates that a variable P option with ±2 bounded variability that is allowed to run for a sufficient length of time, may show results comparable to those expected from a fixed P option.

14.3 Population size sensitivity

The major factor still influencing the performance of the MCGA with respect to population size is the impact of proper P selection. The performance with P of 20, 60, 100, 200 (same as used above), and 600 on the TRIPLEX problem are analyzed in the following figures. The results for each population size are averages of 20 different seeds.

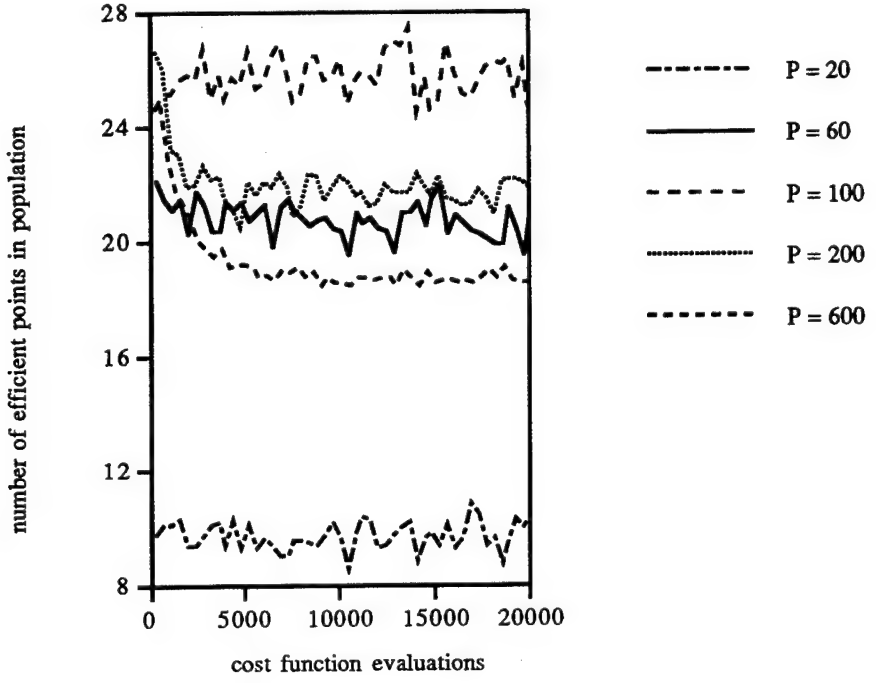


Figure 14-22: Size of **e** comparison for various P on TRIPLEX problem

Figure 14-22 shows that the number of points in **e** does vary with the population size chosen but does not change appreciably throughout a lengthy 20,000 cfe run. There is a definite correlation between the number of efficient points and the population size used, but it is not a direct correlation! For the limited numbers of P attempted, the best value was 100. The number of points in **e** gets better as P increases from 20 to 100, and gets *worse* as P increases from 100 to 600.

Figure 14-23 shows the variance of the distance between members of **e**. The lines shown are a 6th order polynomial curve fit through the averages of the 20 seeds to better illustrate the trends of the data.

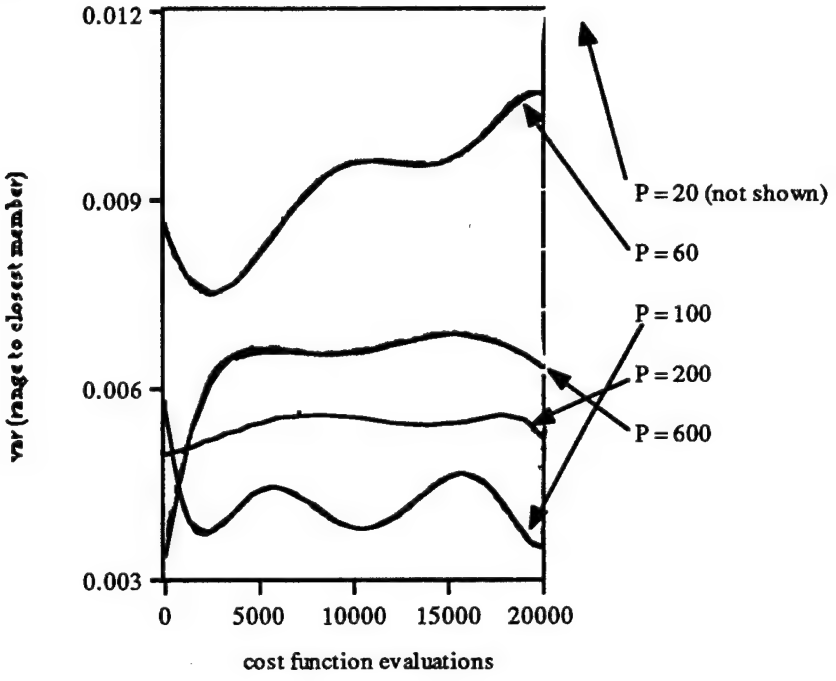


Figure 14-23: **e** range variance for various P on TRIPLEX problem

The 20 member population size is to poor to even show up on the scale used for the figure, while all of the other $P \geq 60$ exhibit good performance. Figure 14-23 shows that while the variance of efficient point spacing is affected by the P chosen, good performance on this measure can be obtained by simply choosing reasonable population sizes. Again, the best performance for the 5 P tested is obtained by a 100 member population size. The impact is actually very significant over P = 60, and somewhat less so over $P \geq 200$.

The quality of the respective populations is best in larger populations as is shown in Figure 14-24. The P = 200 line is the same as in Figure 14-16

where the fixed population size option MCGA outperformed the variable P options. The P=100 line, which shows the best performance in the other two performance criteria exhibits compatible distance measure performance with the variable P options in Figure 14-16.

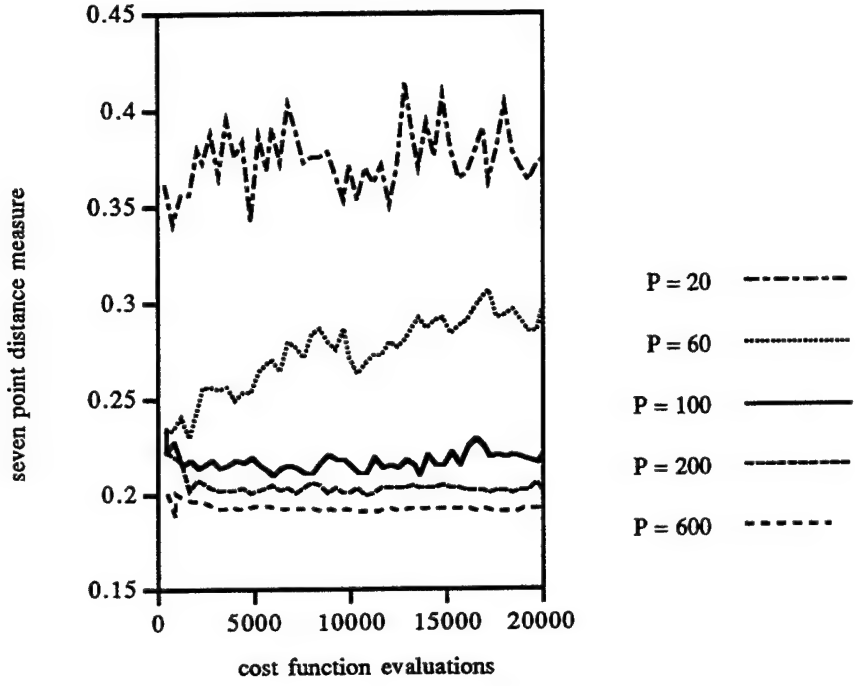


Figure 14-24: Distance measure for various P on TRIPLEX problem

This figure shows that the ability of the MCGA to place its population in proximity of **E** improves as P increases. The improvement, however, is not significant above P = 100. Up to P = 100, however, the distance measure of the population actually increases over time. This fact, coupled with the uncertainty involved in picking P for good performance in all measures, makes the importance of further testing on how to choose the appropriate P very urgent indeed.

The results shown here for the TRIPLEX problem do not allow us to form generalizable impressions about the proper MCGA configuration of population size or population variability to use for a general fault tolerant system design problem. It would appear that in most instances a properly chosen population size used in a fixed P option produces good results. The difficulty in choosing the proper P for a general problem, though, makes the shown viability of P variability an attractive alternative. Additional research on the general performance of these options and possible adaptations of the

growth/decline rules for changing the population size may increase the reliability and effectiveness of this method.

14.4 Effect of Tie Definitions

Another MCGA performance issue addressed in this thesis is the effect of candidate domination "tie definitions". A tie is defined between two candidate members when relative dominance cannot be determined. A tie is broken by performing equivalence class sharing as described in Section 11.3.2. The manner in which ties are defined impacts the relative weights that sharing and dominance play in reproduction. Emphasizing dominance over diversity by defining a tie between candidates that dominate the same fraction of the representative tournament (t_{dom}) set (referred to as T_2) should theoretically result in an efficient set representation with high accuracy and poor spacing variance. On the other hand, defining a tie as whenever only one candidate does not dominate all of the t_{dom} set (referred to as T_1) results in more ties being declared and an emphasis of diversity over dominance that should theoretically result in poor quality, but excellent spacing variance. The ensuing analysis shows the empirical results for each option.

The analysis is performed for both the TRIPLEX and TISS problems, but only results for the TISS problem are shown due to the similarity on the two problems. The MCGA control configuration for this analysis includes an initial P of 600 members, t_{dom} of 0.2, fixed population size, and clone penalization. Each option is run for 10,000 cfe with 20 different random number generator seeds.

Figure 14-25 shows that the number of points in **e** is equal for the two options at the beginning and end of the 10,000 cfe run. T_2 holds more efficient points in its population for most of the run.

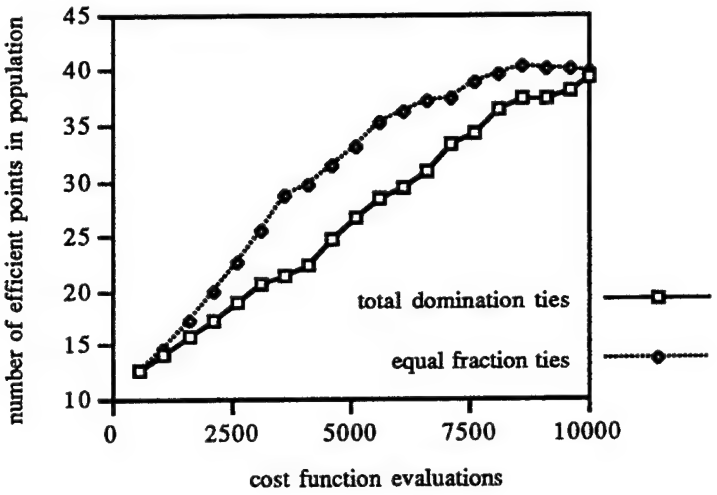


Figure 14-25: Size of **e** comparison of tie definition on TISS problem

Figure 14-26 shows that changing the definition of a tie (and theoretically shifting the dominance/diversity emphasis) does not appear to have a significant impact on spacing variance. Theoretically, T_1 should outperform T_2 in this figure, but that is not evident from the results of either the TRIPLEX (not shown) or TISS problems.

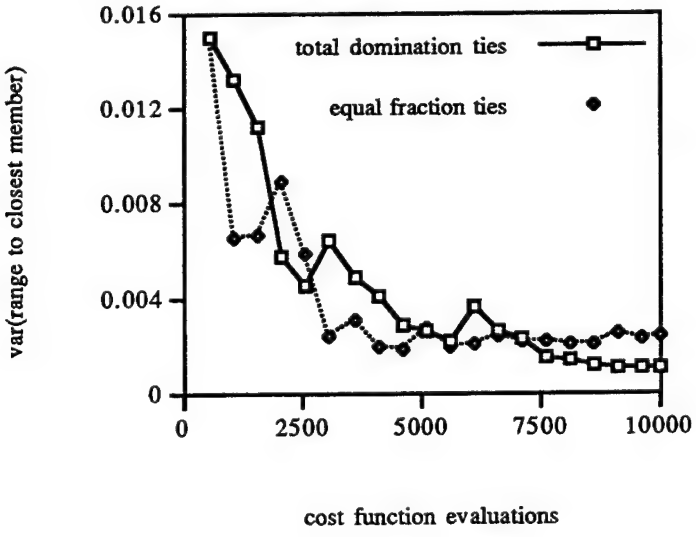


Figure 14-26: **e** range variance of tie definition on TISS problem

Finally, Figure 14-27 shows the accuracy of the two tie definition options.

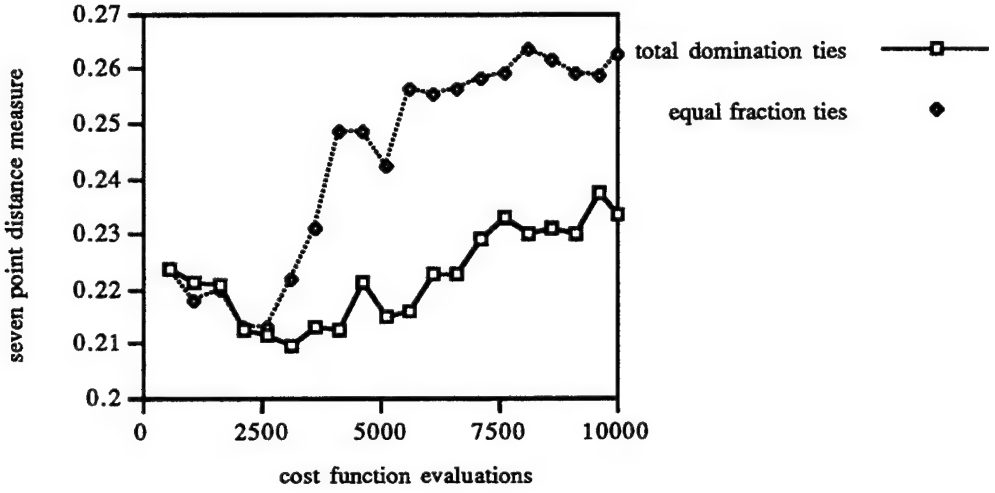


Figure 14-27: Distance measure of tie definition on TISS problem

T_1 has better quality for most of the run, contrary to the hypothesis that it should exhibit reduced accuracy at the cost of maintaining diversity.

The differences between T_1 and T_2 are slight, but the option suggested in [10] of only declaring domination when one (only) candidate dominates the entire t_{dom} set and declaring a tie the rest of the time (T_1) seems to produce better results for the limited test suite of this thesis. The emphasis of dominance in T_2 probably limits the spacing of the population across the full range of attribute values, hindering its ability to approach the seven fixed distance points—thus diversity is important not only to spacing the population equally, but also to reaching different regions of the objective space.

By counting the number of times sharing is performed on the TRIPLEX problem for both tie definitions, a 32% decrease in the number of sharing operations is obtained by using T_2 instead of T_1. The sharing operation can be performed rather rapidly, though, so the small additional computation for diversity emphasis is worth the increase in performance.

15.0 Multicriteria Method Comparison

The constraint method will now be used to generate a representation of the TRIPLEX and TISS problems to provide a comparison for the MCGA. For both problems, a 50 point representation of the efficient set (**E**) is attempted. The two criteria of interest are the unavailability (i.e. 1.0 - reliability) and the system purchase cost (in dollars).

The implementation of the two-dimensional constraint method with the ssga as the underlying single criterion optimization method requires one criterion to be established as a constraint while the other is optimized. In this thesis, the unavailability is constrained (with logarithmic constraint steps between points). A multiplicative fitness penalty function (G) is applied as a cubic of the value of the constraint violation (see Chapter 5 and Section 11.1).

15.1 Triplex problem comparison

The TRIPLEX problem will now be optimized by the constraint method. Table 15-1 gives the input setting for the ssga that influence the constraint method's configuration and performance.

Description	Value
resolution	50 points
Population size	200
crossover rate	0.8
mutation rate	0.001
maximum cfe	5,000
PBL	0.3
G	cubic-decay fitness penalty

Table 15-1: TRIPLEX ssga configuration for constraint method

The configuration of Table 15-1 was used to generate an efficient set representation (**e**) of the TRIPLEX problem. The method performed 186,000 total cost function evaluations, or an average of 3,750 cfe per ssga run. In this attempt, 6 of the 50 initial populations for the constrained ssga problems held all infeasible points for their individually imposed constraint (see Section 11.1), requiring random regeneration. Of the 6 regenerated, 3 were again completely infeasible causing the associated constrained problem to be discarded.

Of the remaining 47 points created, 18 were duplicates of other points (clones) and none were dominated, leaving 29 efficient points for **e**. The statistics of the constraint method's attempt are summarized in Table 15-2.

Description	Value
desired resolution	50
efficient points of solution (**e**)	29
dominated points	0
duplicate points	18
instances where initial population was infeasible	6
instances where second attempt to create an initial population failed and the sub-problem was rejected	3
ssga sub-problem points terminated by PBL = 0.3	11
total cfe of the constraint method	186,000

Table 15-2: Constraint method data for TRIPLEX problem optimization

To facilitate a comparison, a single run of the MCGA has been performed on the TRIPLEX problem. The MCGA was run to 5,000 cfe, with a

crossover rate of 0.80 and a mutation rate of 0.001. The MCGA parameters were set for an initial P of 200 members, P variability of (±2), clone penalization, T_1 tie definitions, and t_{dom} of 0.20.

Table 15-3 shows the performance characteristics of the MCGA and constraint method efficient set representations. The MCGA run terminated with a population size of 90 members.

Performance criterion	Constraint Method	MCGA
cfe	186,000	5,000
range variance	6.737e-4	5.639e-4
distance measure	0.2545	0.2116
efficient points	29	32
duplicate points	18	4
dominated points	0	54

Table 15-3: Performance comparison on TRIPLEX problem

From this table we can observe the impact of transferring even a simple multicriteria problem into a set of single criterion problems to solve it. The constraint method performed 37.2 times more cfe than the MCGA performed to generate comparable results. In fact, the seven point distance measure and the range variance for the constraint method were 16.8% and 16.3% *worse* than MCGA, respectively. Finally, Figure 15-1 shows a comparison of the **e** created by the MCGA and constraint method for the TRIPLEX problem.

Figure 15-1 shows that the two methods agree closely on the **E** of the TRIPLEX problem. The design space for the TRIPLEX problem is very small (1,000 points), so that neither method should struggle in determining a solution. The two methods share most of the points of their respective **e**, but the MCGA has five unique points, two at the lower right corner of the figure, and three in the upper third of the figure. The constraint method found two unique points of its **e**; one located in the center of the figure and the other in the lower right corner.

Three key observations can be made from Figure 15-1. First, the reader is reminded that the actual **E** for the TRIPLEX problem is unknown, but the close agreement between the two methods suggests that both provide good approximations of it. Secondly, allowing the ssga to run for 5,000 cfe on the TRIPLEX problem seemed to be acceptable during single criterion problem

optimization, but the MCGA apparently has the ability to resolve what appears to be an accurate **e** in approximately the same number of cfe as one ssga run of the constraint method. Finally, the difficulty in determining the resolution appropriate for the constraint method beforehand is slightly evident in this problem.

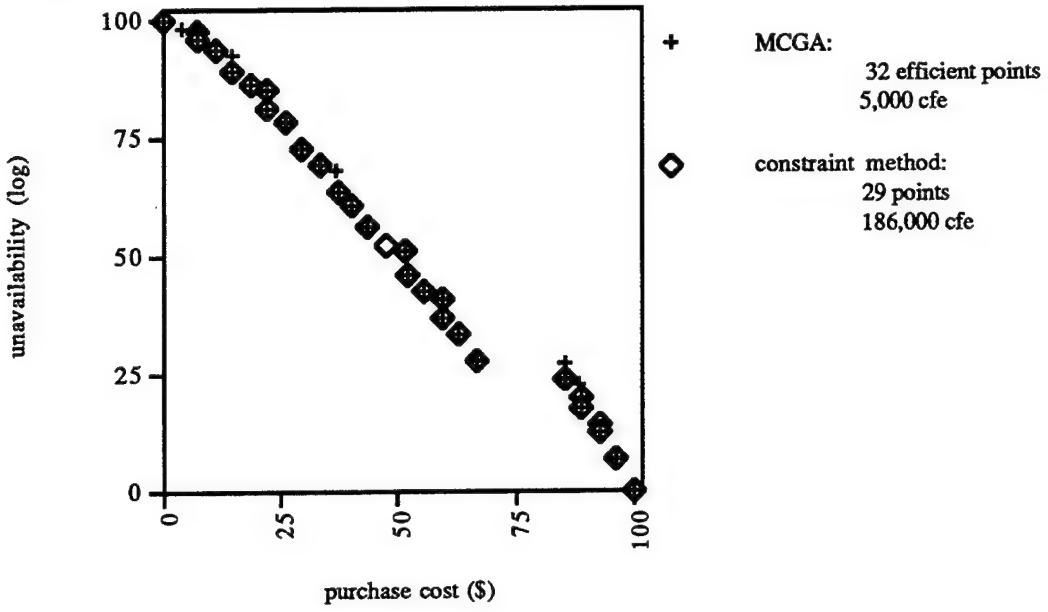


Figure 15-1: MCGA and constraint method on TRIPLEX problem

An arbitrary guess that a 50 point resolution would sufficiently resolve the TRIPLEX problem efficient set causes only 29 efficient points to be located. The method found 18 duplicates (2/3 the number of efficient points), which suggests that the efficient set has been resolved to the closest extent possible. If the resolution was increased, the number of failed ssga runs and duplicate points would increase without gaining a significant number of additional points in **e**. On the other hand, a resolution much smaller than 50 would leave the DM with insufficient **E** information because either (1) the number of points in **e** would be smaller and **E** would be insufficiently resolved, or (2) all ssga runs would generate unique points, causing the DM to ponder whether or not additional resolution could improve **e**. In Figure 15-1, for instance, the tight resolution of the representations and the frequency of duplicates give the DM some assurance that the "jump" of purchase cost when the log of unavailability drops below 25 is imbedded in the problem characteristics and is not a result of insufficient resolution.

15.2 TISS problem comparison

The TISS problem will now be optimized according to the constraint method guidelines given at the beginning of this chapter. Table 15-4 gives the input settings for the ssga that influence the constraint method's configuration and performance:

Description	Value
resolution	50 points
P	600
crossover rate	0.8
mutation rate	0.00033
maximum cfe	10,000
PBL	0.3

Table 15-4: TISS problem ssga configuration for constraint method

The constraint method attempt at the TISS problem required 302,700 cfe, or an average of 6,050 cfe per ssga run. Of the 50 initial populations created, 11 were infeasible and were regenerated. Of those regenerated, all 11 populations were again completely infeasible, causing the sub-problem to be discarded. Therefore, only 39 points were created by the constraint method. One clone and 15 dominated points were created, leaving only 23 efficient points, as shown in Figure 15-2.

Figure 15-2 shows that the 11 rejected ssga problems were tightly constrained for low values of unavailability. This gap in the figure could be a sign of a method deficiency, or could merely indicate that the discrete TISS problem has no efficient points in that region.

The dominated points are close to **e**, showing only that the individual ssga runs terminated close to, but not perfectly at, the constrained problems' optima. This run of the constraint method did find one point in the low range of unavailability—during its unconstrained ssga attempt to locate the individual optima. The point it found actually dominated the point the author originally believed was the bound of **E**. Unfortunately, though, once the constraint was applied, the method was unable to locate another point until unavailability exceeded 20.

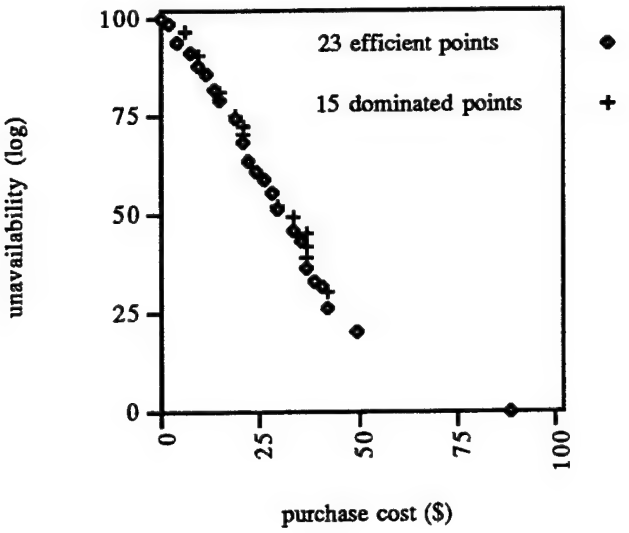


Figure 15-2: TISS problem results for constraint method

The statistics of the constraint method attempt of the TISS problem are summarized in Table 15-5.

Description	Value
desired resolution	50
efficient points of solution	23
dominated points	15
duplicate points	1
instances where initial population was infeasible	11
instances where second attempt to create an initial population failed and the sub-problem was rejected	11
points terminated by PBL = 0.3	27
total cfe of constraint method	302,700

Table 15-5: Constraint method data for TISS problem optimization

To facilitate a comparison, a single run of the MCGA has been performed on the TISS problem. The MCGA was run to 10,000 cfe, with a crossover rate of 0.80 and a mutation rate of 0.00033. The MCGA parameters were set for an initial P of 600 members, P variability of (±2), clone penalization, T_1 tie definitions, and a t_{dom} of 0.20. The MCGA run terminated with a population size of 350 members. Table 15-6 shows the performance characteristics of the MCGA and constraint method efficient set representations.

performance criterion	constraint method	MCGA
cfe	302,700	10,000
range variance	1.318e-2	1.577e-4
distance measure	0.2791	0.2786
efficient points	23	33
duplicate points	1	7
dominated points	15	310

Table 15-6: Constraint method performance data on TISS problem

From this table we observe a marked advantage in using the MCGA. Though its representation (the final ga population) contained 310 dominated and 7 duplicate points, filtering those points out leaves 33 points in **e** compared to only 23 in the constraint method. The distance measures of the methods only differ by 0.2%, but the resolution point spacing (range variance) is nearly 2 orders of magnitude better in the MCGA.

The actual **e** for the MCGA run and the constraint method run are compared in Figure 15-3. It shows that again, as in the TRIPLEX problem, the two methods generally agree on the location of **E**. Unlike the TRIPLEX problem, however, where the design space is very small and the agreement is expected to be close, the large discrete design space of the TISS problem is more difficult for both methods. The best comparison of the two methods can be made by looking at the low unavailability range of Figure 15-3. The constraint method found the lowest unavailability value known using an unconstrained ssga problem, but is unable to generate another point until unavailability increases to over 20. The MCGA, in comparison, does not locate the very lower bound of the efficient set, but it is able to locate points and spread them nicely to values of unavailability of about 12. The most obvious, and significant difference between the methods, though, is the number of cfe each performed. In this test, the constraint method performs 30 times as many cfe as the MCGA!

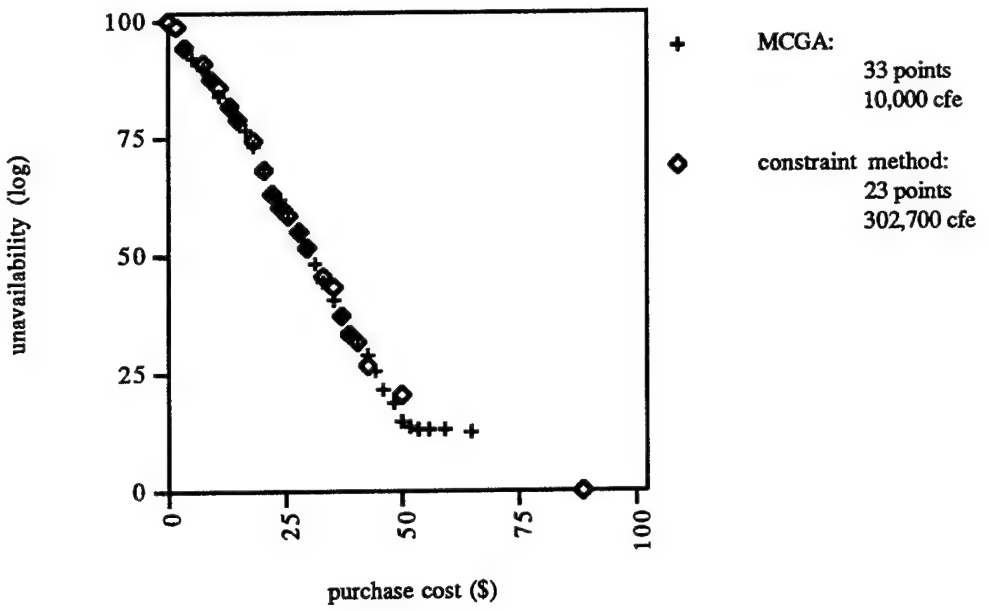


Figure 15-3: MCGA and constraint method comparison on TISS problem

From this limited comparison using two discrete, two-criteria fault tolerant system design problems, we can conclude that the MCGA is capable of generating an excellent **e** in a low number of cfe when compared to a method that require many single-criterion sub-problems to be solved. The MCGA spreads its representation very evenly as shown by the low range variance values produced, but it appears to experience some difficulty in locating those points at the far limits of the efficient set. Some additional work on the mechanisms of the MCGA may be able to aid the expansion of its solution to cover a greater range of the efficient set.

Though the results of Chapter 14 show rather conclusively that using a fixed population size in the MCGA will provide the best performance, a variable P has been used in this comparison to show that even at its slightly degraded level of performance, the MCGA outperforms the constraint method handily. The (±2) variability is also used for a reason that affects the constraint method as well—a lack of understanding beforehand of what the proper resolution should be. In both methods, choosing the resolution too small affects the representation, while choosing it too large requires excessive cfe. Allowing the MCGA to choose its own population size removes that burden from the DM.

The final issue of contention necessary to compare the MCGA and constraint methods is their ability to handle problems with more than two

criteria. As described in Section 11.1, the constraint method is severely limited by such problems. The MCGA, on the other hand, adapts readily to many criteria because it always chooses members on the basis of dominance, which is defined identically for any number of criteria. This thesis has not investigated problems of more than two criteria, but the MCGA suffers no theoretical limitations of larger dimensioned problems.

16.0 Summary of Multicriteria Design

Fault tolerant system optimization normally requires extensive and usually subtle tradeoffs between factors such as component quality, reconfiguration strategies, level of redundancy, and operational policies. Optimization strategies must incorporate the conflicting effects of such constraints as performance specifications, reliability goals, and size and weight in order to design to minimize cost. Whenever possible, the designer attempts to combine all criteria of interest into a single cost function. However, criteria are not always commensurate, requiring the introduction of multicriteria optimization techniques.

Several types of multicriteria optimization can be performed depending on the level of involvement the DM wishes to have in the optimization process, and the computational constraints involved. This thesis looks solely at generating methods of multicriteria optimization where a representation of the problem's efficient set is generated so that the DM not only has a number of potential solutions from which to choose, but she is also given a greater understanding of the relationship between the criteria.

The most common generating techniques in use are the ε-constraint and weighting methods. A form of the ε-constraint method is created for this thesis to allow optimization of two-criteria problems. The single criterion ssga is used as the underlying optimization method to capitalize on the robustness of the method demonstrated earlier in this thesis and to allow the optimization of mixed and discrete parameter fault tolerant problems. The constraint framework using a multiplicative fitness penalty function described in Chapter

5 is included that allows the ga to scale the penalty automatically . The ability of the ga to effectively deal with function constraints in this manner is demonstrated by its good performance in Chapter 15.

The main thrust of the latter half of this thesis is the examination of a genetic algorithm that uses dominance as its string selection criterion to operate on multicriteria fault tolerant design problem directly. The Multicriteria Genetic Algorithm (MCGA) represents a fresh approach for multicriteria optimization. Its remarkable effectiveness may generate revolution in optimal design. It promises to revolutionize the ability of a DM to analyze the complex tradeoffs of non-commensurate criteria and produce more versatile, satisfactory designs.

The MCGA is an unproved capability at the time of this writing. The ability of the ga in general to perform multicriteria optimization has only had limited analysis. Unlike the single criterion application of the ga, where the basic framework and performance of the ga has been worked out and debated for several years, multicriteria ga implementations are still in their infancy. Therefore, this thesis not only had to research whether ga's can do multicriteria optimization of fault tolerant system design better than other methods, it also needed to investigate the basic viability of ga's in multicriteria optimization and the form the ga should take.

This thesis explored the effects clones, population variability, and candidate member tie definitions have on MCGA performance. A set of performance criteria for evaluating multicriteria methods has been created and used to compare the performance of various MCGA configurations. Clones significantly affect performance, and the best method of dealing with them is to allow their presence in the population, but with heavy penalties assigned to their reproductive ability.

The effect of population size variability has been examined to determine whether the MCGA is capable of varying its own population size for optimal performance because of the difficulty in determining a proper fixed population size. The results show that the advantage of population size variability inclusion depends on the user's confidence in the fixed population size chosen. A properly sized fixed population outperforms a variable population size, while the performance of a variable population size MCGA outperforms fixed population attempts with poorly chosen population sizes.

Tournament selection with domination as the selection criterion is used as the basis of selecting parents for mating in the MCGA. The manner in which domination is defined dictated that a tie between candidate solutions has to be decided by equivalence class sharing to spread the population along the efficient front. The definition of ties is explored for its effect on the conflict between domination and diversity in the MCGA population. This research reveals that on the test problems examined, defining ties as when one (only) candidate does not dominate the entire tournament set provides the best MCGA performance. This result shows that diversity maintenance via sharing is crucial to optimal MCGA performance.

A small comparison of the MCGA with the constraint method shows that both methods are effective at providing accurate representations of the multicriteria efficient set. The MCGA proves to be superior for three main reasons: (1) the difficulty involved in using the constraint method for three or more criteria, (2) the much larger numbers of cost functions required to optimize many ssga single-criterion problems for the constraint method representation compared to the MCGA which optimizes directly on the efficient set representation, and (3) the uncertainty involved in choosing a proper efficient set resolution from the constraint method that properly covers the efficient set without creating excessive computational effort. The MCGA resolution is limited only by the population size if population variability is not permitted or only by the effectiveness of the method if population size variability is allowed.

The performance of the MCGA can be expected to surpass other generating methods more as the model complexity, number of criteria, and discrete nature of problem increase. In all three cases, a genetic algorithm that uses domination as its selection criterion continues retains its robustness and shows a superior ability to generate a highly resolved, quality efficient set representation. The results of this thesis suggest that the MCGA is an effective means of optimizing multicriteria fault tolerant system design problems easily, quickly, and accurately—the design community needs to take a long, hard look at how to best capitalize on the strength of the MCGA.

17.0 Suggestions for Further Work

17.1 Single Criterion

The field of genetic programming is expanding at an exponential rate. The massive influx of effort and literature in this field (of which this work is included) reflects the exciting potential so much of the scientific community sees in the ga. The biannual proceeding of the International Conference on Genetic Algorithms (ICGA) provides a sampling of the astounding advances currently being made. The Internet access provided by David Goldberg and his associates at the University of Illinois Genetic Algorithm Laboratory (IlliGAL) and the user group comp.ai.genetic are just two of the means by which the ga community is communicating to speed the spread of progress in the field.

Further work on the applicability of ga's to fault tolerant system design could incorporate many of the advancements being pursued around the world due to the general, problem-independent nature of the ga. Some of the potentially useful features located recently include introns between parameters on a string and incest prevention in mating.

Those specific items that revealed themselves over the course of this research as worthy of additional investigation include:

Disallowed Parameters

The direct effect disallowed parameters have on genetic algorithm convergence and exploration has not been rigorously pursued in this thesis.

The impact of "wasted" space on the binary representation of parameters was analyzed in Section 8.5, but no conclusions were drawn from the investigation. The fact that the ssga is more significantly impacted by disallowed parameters than the tga is substantiated by this investigation, but the reason for this conclusion has not been determined.

Clones

As was stated in Chapter 4 in the discussion of the ssga used in this thesis, the concept of "steady-state without duplicates" is not used because (1) the ssga used here is an independent development that did not take the work of others into account in its original configuration and (2) the use of "steady-state without duplicates" eliminates the ability to determine convergence based on the principles of Chapter 6. "Steady-state without duplicates", as described in [3], discards children that are duplicates of current members of the population. This keeps the entire population unique. The benefits cited by Davis are the much more efficient use of the reproductive cycle and greater population diversity.

Additional work on the ssga could make use of "steady-state without duplicates" where diversity and computational efficiency is considered more important than the ability to detect convergence.

Convergence

The efforts to define ga convergence in this thesis (see Chapter 6) provide a great deal of insight into the behavior of the ssga as it is presently configured. The "product of the bit likeness" (PBL) figure of merit shows that convergent behavior can be observed for the ssga and an effective termination criterion can be developed. Additional research that looks at some form of the PBL gradient to determine when the sharp rise occurs would stand to improve the robustness of this termination criterion.

Function Constraints:

The ability of the ga to assign its own penalty function scaling is a concept the author believes is original to this thesis. The theoretical basis has not been rigorously developed and general applicability to a wide set of function

constraints has not been investigated. The benefits of this novel approach have only been tested in the multicriteria context of Section 11.1. Additional work in this area should build an analytical and empirical framework for the viability of this general penalty function approach.

Parallelization

That the ga is highly suited for parallel computing is rarely refuted. The efforts of this thesis are limited to serial computing applications. However, the framework established could be adapted to parallel configurations of either:
1) distributed processing among linked workstations or personal computers
2) parallel computing hardware

In [17], the authors reference a population size study by G. G. Robertson (Proceedings of the 5th International Conference on Machine Learning, 1988), which found that performance using a parallel computing system monotonically increases with population size. This is not surprising when there is no real "cost" for larger populations. In serial machines mainly available today, there is a fixed cost increment for each population member; a real tradeoff between population size and "performance" exists.

Coupling the ssga, which provides optimal performance when computational limitations are eased and population sizes are increased, and parallel configurations will undoubtedly create a very competitive optimization framework.

17.2 Multicriteria

Tournament set size (t_{dom})

As presented in Section 12.4, the tournament set size t_{dom} is one of the four MCGA performance parameters that must be set correctly for optimal MCGA performance. This thesis used a fixed t_{dom} of 0.2, or 20 percent of the current population size. The appropriateness of this choice has not been verified. Reference [10] includes a brief look at the general setting of t_{dom}, but an in-depth analysis of the appropriate tournament size for multicriteria genetic algorithm optimization is critical for the furtherance of this ga application.

Convergence testing

The ability to determine satisfactory ga convergence and terminate the algorithm is especially difficult in the MCGA because the algorithm is converging to a population of points that represent an unknown quantity. The development of a means of termination based on MCGA convergence may be linked to convergence of the ssga with "steady-state without duplicates" applied.

Representing the efficient set bounds

The MCGA is shown in Chapter 15 to have some amount of difficulty reaching the boundaries of the efficient set. The push towards dominance seems to be greatest where tradeoffs between criteria are the smallest. Unfortunately, DM may need information on the limitations-of-possibility, which requires some assurance that the efficient set representation includes the actual bounds. The MCGA, as it is presently formulated, needs to be augmented with additional operators to enhance exploration toward the edges of the efficient set or will have to used in conjunction with another method.

One option that has not be tested is to run individual ssga runs for each criterion to determine an approximation of the bounds, and then to include those points into the MCGA population with a multicriteria form of elitism to enhance their ability to produce offspring.

Constraint method

Though the constraint method has been shown to be generally inferior to the MCGA, additional work in the area of multicriteria optimization will always need sound competitive methods to compare performance against. A few simple configuration changes of the constraint method used in this thesis should greatly improve the competitiveness of the method:

1) If a good initial population is generated, each subsequent ssga run for the constraint method could start with that initial population, thus saving a great deal number of cfe. In fact, the effort required would be equal to the population size times (resolution - 1).

2) The constraint method has the most difficulty finding points when the underlying single-criterion problem is tightly constrained. As such, better performance can be attained in two-criteria problems by switching the criterion and the constraint with each other at some mid-point of operation. In this way, both edges of the efficient set are located by sub-problems with larger feasibility regions.

Appendix A: The Markov Modeling Method

A.1 Background

Markov modeling techniques provide a systematic means of investigating system reliability for large, complex systems and determining life cycle (time dependent) system costs. They permit the inclusion of sequence dependent events such as repairs in a natural fashion. One of the most powerful aspects of Markov models is their ability to permit simplifying approximations to be made and to provide means to obtain bounds on these approximations. The basic concepts of Markov modeling are best introduced by simple, but representative examples. These examples clearly point out the general flexibility as well as the main drawback of the method, particularly the rapid expansion of the state space. Techniques used to reduce the state space to manageable proportions, without compromising the quality of the analysis, are described in detail in [1].

A.2 Single-Component System

Figure A-1 shows a single-component system. The first step in modeling the reliability of this system is to determine what the system requires to be in an operational state. This single-component system has a trivial operational requirement: it is operational if the single component, A, has not failed. (Conversely, the system is failed if component A has failed). While this step is simple for this system, it is often one of the most complicated steps in modeling a complex system, characterized by many operational states and subtle interactions among components.

Figure A-1: Single component system block diagram

Given the system operational requirements, the next step is to construct Markov model states. A *state* represents a unique configuration of failed and operational elements, sometimes distinguished by the sequence of

the failures that led to it. Figure A-2 shows the Markov model for the one-element system. In general, a model is generated by first creating state 1, the state where there are no failed components in the system. The various transitions out of state 1 represent failures of the system components, accounted for individually or in groups. In this case there is only one component, thus a transition denoted λ is created leading to state 2. This state represents this system when component A is failed. Noting the operational requirements for this system, state 2 is labeled as a system failure. Since there is only one component in the system and its failure has been accounted for, the Markov model is complete.

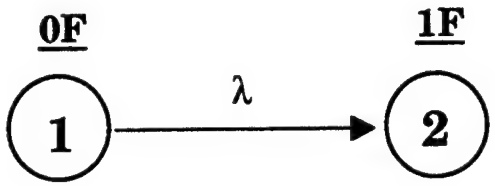


Figure A-2: Single component system Markov model

This system's reliability is just the probability, as a function of time, of being in state 1. Actually, there is a probability associated with each state. For example, at time zero the probability of being in state 1 (no failures) is 1 (or 100%) and the probability of being in state 2, or any other state, is 0. Parameter λ on the transition in the model not only indicates that component A has failed along this transition, but that the component's failure rate is λ failures per hour. Throughout our discussion, it will be assumed that all failure rates are constant in time. To obtain the system reliability as well as other state probabilities of interest as a function of time, we need to track the probability "flowing" out of state 1 into state 2. Probability flow is the product of the transition rate and the state probability for the state at the origin of the transition. Thus, a state with zero probability has no probability flowing out of it, a state with no exiting transitions has no flow out, and a state with probability equal to 1 and an exiting transition rate of λ has an instantaneous flow out equal to λ. The rate of change of each probability is then given by the net probability flow into the corresponding state. A Markov model is thus

mathematically described by a set of differential equations governing the evolution in time of the probabilities of being in each state.

Using the definition of the probability flows, the following equations are obtained for the Markov model shown in Figure A-2:

$$\frac{dP_1(t)}{dt} = -\lambda P_1(t) \tag{1}$$

$$\frac{dP_2(t)}{dt} = \lambda P_1(t) \tag{2}$$

These equations, representing the rate of changes in each state variable (P_1 and P_2), are called state equations. Equation (1) shows that the rate of change in probability for state 1 is the exiting transition rate λ times the probability of being in state 1. The minus sign indicates that the transition is out of the state and, therefore, reduces the probability of being in state 1. Equation (2) is interpreted similarly. Note that the flow is *into* state 2; the positive term indicates an entering transition which increases the probability in state 2. Also, the flow into state 2 is the rate λ times the probability of *state 1*; the flow on this transition is due to state 1, the origin of the transition. Equations (1) and (2), along with the initial condition of the state probabilities, $P_1(0) = 1$ and $P_2(0) = 0$, provide a complete description of the system's reliability. Markov models have the property that a flow leaving one state enters another, as shown in Equations (1) and (2). Hence, the *total* system probability does not change as the system evolves. This fundamental property is called conservation of probability. The sum of all the system's state probabilities is always equal to 1.

There are many ways of solving Equations (1) and (2) in closed form, such as standard integration or Laplace transform. Using any convenient technique and recalling that the failure rate λ is constant, yields the solution:

$$P_1(t) = e^{-\lambda t} \tag{3}$$

$$P_2(t) = 1 - e^{-\lambda t} \tag{4}$$

State 1 starts with a probability of 1 and decays exponentially toward 0, while state 2 has a probability initially at 0 which grows toward 1. Notice that

the sum of the two state probabilities is 1 at all times, thus indicating the conservation of probability.

Two Component System with Repairs

Figure A-3 shows a two-component system where the components are connected in parallel. The requirement for system operation is that at least one of the two components is working. These components can be repaired when they are failed.

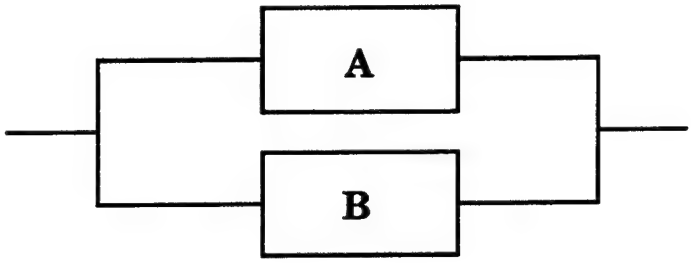


Figure A-3: Two component system block diagram

The Markov model of this system is shown in Figure A-4. State 1 represents the no-failure configuration. Possible events when in this state are that component A can fail or component B can fail. These two possibilities are captured in the transitions leaving state 1 with the rates λ_1 and λ_2 respectively. State 2 represents component A failed and B working. Possible events leading out of state 2 are that component B may fail (exiting transition λ_2) or that component A may be repaired (exiting transition μ_1). Here, μ_1 stands for the repair rate for component A.

The failure of component B leads to state 4, while the repair of component A leads back to state 1, returning the system to the no-failure state. Similarly, the exiting transitions for state 3, the B failed/A working state, are a failure of component A (transition λ_1 going to state 5) and a repair of component B (transition μ_2 going back to state 1). Notice that repairs, which are sequence-dependent events (since they can only be performed *after* a component has failed), are easily included in the model.

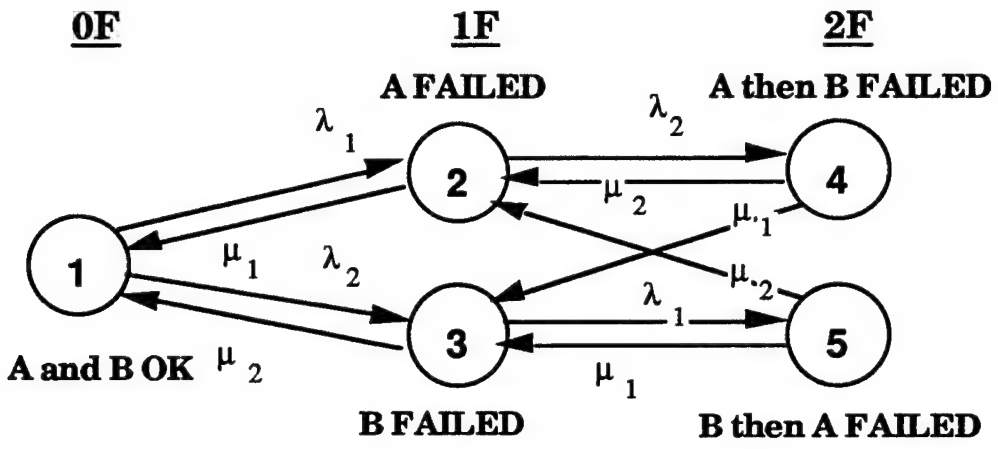


Figure A-4: Two component system Markov model

States 4 and 5 represent system failure, being distinguished only by the sequence of events leading to the loss of both components. States 1, 2, and 3 represent the system in an operational configuration. If one was concerned with degraded operational modes, such as operating without a backup, then this model could also provide that information by giving the probability of states 2 and 3 independent of state 1.

States 4 and 5 both represent system configurations where components A and B are failed. However, in state 4 component A failed first and in state 5 component B failed first. In both of these states, the possible events are the repair of A (transition μ_1 leading to state 3) and the repair of B (transition μ_2 leading to state 2). Since the possible actions taken and their consequences, i.e., the destination states, are the same in states 4 and 5, these states may be lumped together if the order-of-failure distinction is not needed in the analysis. The resulting model is shown in Figure A-5. This simplification is referred to as exact aggregation of states and introduces no approximations. It is useful in systems where there are many identical components each with an identical impact on the system operation.

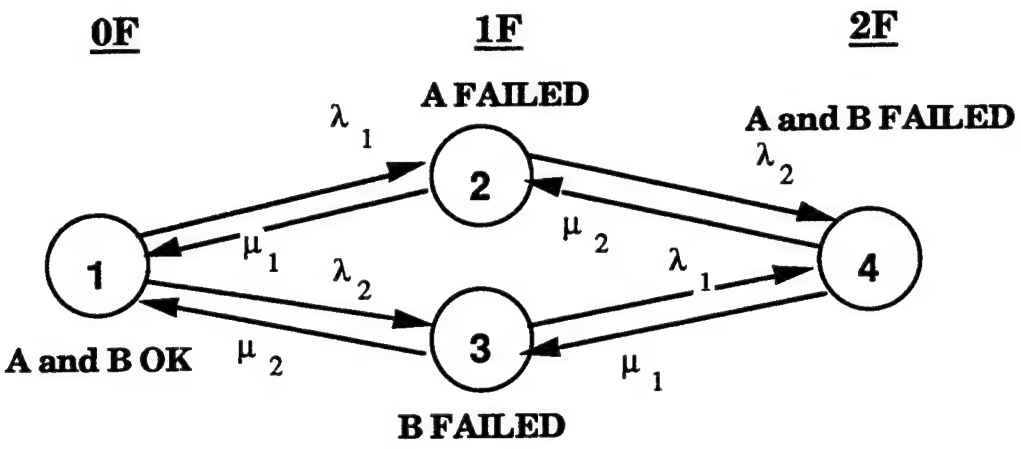


Figure A-5: Aggregated two component system Markov model

The state equations for the model in Figure A-5 are obtained by inspection of the model diagram and applying the rule for determining flows. The state equations are:

$$\frac{dP_1(t)}{dt} = -(\lambda_1+\lambda_2)P_1(t)+\mu_1P_2(t)+\mu_2P_3(t) \tag{5}$$

$$\frac{dP_2(t)}{dt} = \lambda_1P_1(t)-(\lambda_2+\mu_1)P_2(t)+\mu_2P_4(t) \tag{6}$$

$$\frac{dP_3(t)}{dt} = \lambda_2P_1(t)-(\lambda_1+\mu_2)P_3(t)+\mu_1P_4(t) \tag{7}$$

$$\frac{dP_4(t)}{dt} = \lambda_2P_2(t)+\lambda_1P_3(t)-(\mu_1+\mu_2)P_4(t) \tag{8}$$

Note that all flows leaving a state (negative terms) appear as a flow entering a state (positive terms), thus indeed probability is conserved. Equations (5) through (8), together with the initial condition that state 1 has a probability of 1 and all other states have probabilities of 0 at time = 0, provide a complete description of the system. Figure 1-2 is the same as that in Figure A-5, except that the repair from state 4 is assumed to always be a complete system repair that takes the system back to state 1.

All systems reach a point where the state probabilities are no longer changing. In the example of the single-component system this situation occurred when all of the probability was in state 2 and none was in state 1. This is common for systems without repair. After a long period of time most states have probabilities of 0 and only a few states, called trapping states, have probabilities that are between 0 and 1. In the case of systems with repairs, however, when they reach steady state, all their state probabilities may be between 0 and 1. This comes about because a balance is reached between the flows leaving and those entering the states. For example, when the flow leaving state 1 in Figure A-5 equals the flow entering state 1, its probability no longer changes. This occurs when the probabilities of states 1, 2, and 3 obtain values such that the flows are in balance. Equation (5) shows that this balance is obtained when $dP_1(t) / dt = 0$. Similarly, when the derivatives of all state probabilities are equal to 0, the system has come to its steady state.

The steady state is an important characteristic of fault tolerant system analysis. After an initial transient phase, most systems will operate continuously over a much longer period of time. Although various components fail and are repaired as the system evolves, the probabilities of the various system states have come to steady state. Therefore, the system analysis is, in fact, an analysis of the system operating at steady state. For the completeness of this introduction however, we will briefly discuss the time-dependent problem.

The closed-form solution of equations (5) through (8) for this two-component system, as is true of most systems with repairs, is rather complex and not particularly enlightening. It is more common to solve such system models numerically.

First, the system equations are written in matrix form:

$$\frac{dP(t)}{dt} = \begin{bmatrix} -(\lambda_1+\lambda_2) & \mu_1 & \mu_2 & 0 \\ \lambda_1 & -(\lambda_2+\mu_1) & 0 & \mu_2 \\ \lambda_2 & 0 & -(\lambda_1+\mu_2) & \mu_1 \\ 0 & \lambda_2 & \lambda_1 & -(\mu_1+\mu_2) \end{bmatrix} P(t) \qquad (9)$$

where the state vector is:

$$P(t) = [P_1(t), P_2(t), P_3(t), P_4(t)]^T \qquad (10)$$

Notice that the columns of the matrix add to zero. This represents the flow conservation property in the system: all flows leaving a state must enter another state. The matrix equation may be written more concisely as:

$$\frac{dP(t)}{dt} = \mathbf{A}P(t) \qquad (11)$$

Equation (11) is the continuous-time representation of the Markov model. Matrix **A** is the continuous-time transition matrix. While there are many ways of numerically integrating this equation, the one shown here is straightforward and adequate in many situations. The derivative is approximated over a discrete time step Δt by:

$$[P(t+\Delta t) - P(t)] / \Delta t = \mathbf{A}\,P(t)$$

Multiplying each side by Δt and moving the state vector P(t) to the right-hand side gives:

$$P(t+\Delta t) = [\mathbf{I} + \mathbf{A}\,\Delta t]\,P(t)$$

where matrix **I** is the identity matrix. The term in brackets may be relabeled as matrix **M**:

$$P(t+\Delta t) = \mathbf{M}\,P(t) \qquad (12)$$

M is the discrete-time transition matrix. The above approximation, Equation (12), is called Forward (or Explicit) Euler integration.

Equation (12) represents a recursive solution for the Markov model. Given the system's initial condition, P(0), it is possible to use this equation to propagate the state probability in time:

$$P(\Delta t) = \mathbf{M}\, P(0)$$

$$P(2\Delta t) = \mathbf{M}\, P(\Delta t)$$

$$P(3\Delta t) = \mathbf{M}\, P(2\Delta t)$$

$$P(4\Delta t) = \mathbf{M}\, P(3\Delta t)$$

$$\vdots$$

$$P(n\Delta t) = \mathbf{M}\, P[(n-1)\Delta t] = \mathbf{M}^n\, P(0)$$

The above procedure gives the state probabilities as a function of time from time = 0 to time = $n\Delta t$. It may also be viewed as an iterative solution of the steady-state problem, i.e., $A\,P(t) = 0$, if continued until the state probabilities no longer change.

A few remarks need to be made concerning this solution procedure. First, Δt must be judiciously selected such that the integration is stable, has the desired accuracy and produces meaningful probabilities, i.e., between 0 and 1. Second, in performing these calculations on a computer, special care must be taken lest round-off errors destroy the solution. Finally, a faster version of this integration scheme, taking advantage of the fact that **M** is time-invariant, may be constructed, based on a very efficient technique to evaluate powers of a matrix.

Appendix B: The Branch and Bound Method

The advantages and disadvantages of genetic algorithms cannot be determined without comparing them against other, proven discrete-parameter optimization techniques. Comparing the different methods can provide an assessment of each based on such figures of merit as real-time operating duration, consistency and reliability of convergence, and ease of use.

From the Society for Industrial and Applied Mathematics (SIAM) course notes tutorial on Numerical Optimization Algorithms and Software [19], page 48: "In many applications the solution of an optimization problem only makes sense if certain of the unknowns . . . are integers. . . Although a number of algorithms have been proposed for [the integer programming problem], the branch-and-bound technique is used in almost all the software we collected. The technique has proven to be efficient on practical problems, and it has the added advantage that it solves 'continuous' linear programs . . . as sub-problems . . ."

The branch and bound optimization method provides a means of solving a set of constrained continuous problems in order to find a suitable discrete or integer solution. Branch and bound has two phases: (1) partition the sample space of solutions into mutually exclusive and completely exhaustive sets by a specified decision algorithm (branch); (2) create upper and lower bounds over the objective functions in these sets (bound). These phases are repeated until the solution is better than the bounds on all the unexplored sets (optimal).

Branch and bound is actually not an optimization method in itself. It is a record keeping algorithm used to track the progress of a subordinate optimization method. The branch and bound capability of DOME uses the Down-hill Simplex continuous parameter optimization method to solve the stack of potential solutions (branches). As the method progresses, it further constrains the discrete parameters for each sub problem on its stack of potential solutions. Because of its underlying continuous nature, branch and bound can not only adjust itself to mixed continuous/discrete parameter problems, it's abilities are improved when the discrete constraints are relaxed. This stands in contrast to the genetic algorithm, which fairs better as the solution space is contracted.

The Mechanics of the Method

The branch and bound discrete optimization method is essentially a means of finding a discrete optimization solution through the repeated solution of continuous optimization sub-problems. Each sub-problem is a more constrained problem than the previously solved one.

Formally, the mixed integer optimization problem is to find the solution $\mathbf{x}^*$ to the problem

$$P_D\text{: Optimize } f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in R\,,\ x_d \text{ is discrete valued (for all) } d \in D \tag{1}$$

where D is the set of discrete variables, and R is the feasible region of the continuous problem

$$P\text{: Optimize } f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in R \tag{2}$$

Next, assume that $\mathbf{x}'$ is the minimizer of P_D and is found. If $\mathbf{x}'$ is a discrete feasible solution, then the problem P_D is solved. If not, then there exists a $d \in D$ for which x'_d is not discretely valued; that is, one of the components x'_d of the vector $\mathbf{x}'$ is not an acceptable discrete value. In this situation, we *branch* on variable x_d in problem P to create two sub problems

$$P\text{-: minimize } f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in R\,,\ x_d \le [x'_d]- \tag{3}$$

and

$$P\text{+: minimize } f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in R\,,\ x_d \ge [x'_d]+ \tag{4}$$

where $[x'_d]-$ means the largest discrete value not greater than x'_d, and $[x'_d]+$ means the smallest discrete value not less than x'_d. Essentially, we now have two problems with smaller feasible regions since x_d is now bounded in both problems. Note that $\mathbf{x}^*$ is the feasible solution in either P- or P+, but not both.

The branching process is repeated by branching on P- and P+ and other sub-problems, resulting in a tree structure of optimization problems like that

of Figure B-1. The optimization of a sub problem will result in one of three situations: (1) the problem has no feasible solution and can be discarded; (2) the problem has a feasible solution that is not discrete-valued, requiring a further branching; or (3) the problem has a solution that is discrete-valued and may be the optimal solution.

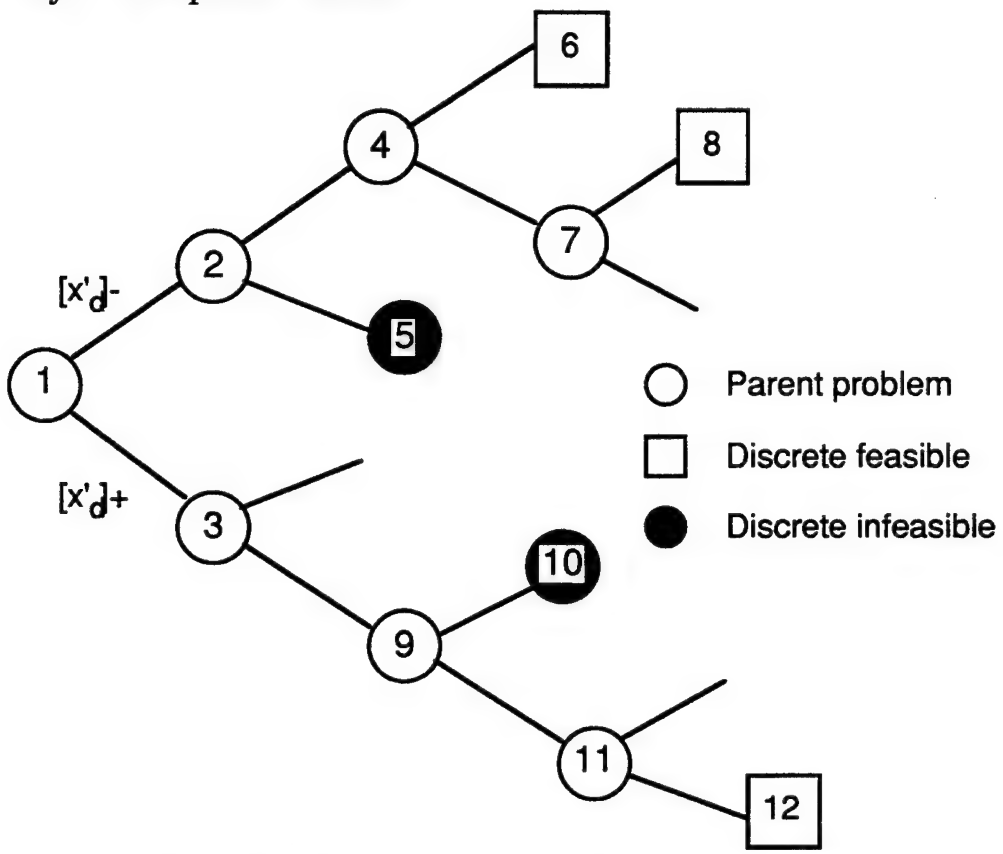


Figure B-1: Tree structure for Branch and Bound optimization

A drawback with this approach is that the number of nodes in the tree grows exponentially with the number of variables and may not even be finite. Thus, an exhaustive search is not efficient. The branch and bound method attempts, however, to find the solution through only a partial search of the tree. Note that the optimal objective function of P must be less than or equal to the optimal objective functions of P- and P+, otherwise the optimal solution to P was not found originally. Thus, the optimal objective function of the parent problem is a lower bound on all its sub-problems. This relationship can be used to reject certain branches of the tree if they cannot improve the current best solution. For example, suppose some, but not all, of the problems within the tree have been solved. Let the best discrete-valued solution have a

cost function value of f'. Suppose that other branches in the tree have lower bounds on the objective function f, such that $f > f'$. These branches can be removed from the tree since none of the solutions will improve on the current best. This principle allows a limited search of the tree to be effective.

There are two remaining issues to be addressed: first, which sub-problem should be solved next; and second, on which variable should the branch be made. The DOME program uses a stack to keep track of the unsolved problems, each with a lower bound on the minimum of the objective function. The effect of the stack method is to follow a single path deep into the tree to find a discrete feasible solution. Then the algorithm works back, creating more sub-trees or rejecting problems. The algorithm branches on the variable that has the largest absolute error from its nearest acceptable bin value. For example, for an integer problem, if $x_1 = 3.2$ and $x_2 = 6.4$, the algorithm will use x_2 as the branch variable. The problem placed on the top on the stack is the one with the bound closest to the branched variable. The other sub-problem is placed second on the stack. For example, for an integer-valued problem, if the branched variable $x_2=6.4$, the top problem on the stack will have an *upper* bound on x_2 of 6, and the second problem on the stack will have a *lower* bound on x_2 of 7. In this way, the branch most likely to contain the optimum is tried first (the branch with the upper bound on x_2 of 6 in this instance) to try to trim the tree as quickly as possible.

The stack method of Branch and Bound optimization has proven in practice to be quite effective for the type of problems solved by the DOME program. The relative value of the Branch and Bound method using the stack method will be determined by its ability to find a discrete feasible solution in a reasonably small number of sub-problems, which is important when the number of iterations is limited, and how consistently the method is capable of finding a *quality* solution.

Though most any optimization tool can be used as the underlying method with Branch and Bound, the Down-hill Simplex method of [21] is used in DOME. A brief discussion of this method follows in the next section.

Appendix C: Down-hill Simplex

The Down-hill Simplex method [21] has very robust characteristics and can converge to a solution reliably in linear and non-linear programming problems. It also has the advantage of not requiring any derivative information of the cost function. This allows the branch and bound method to operate on the same cost functions as the genetic algorithm and represents the major reason why it was chosen for inclusion in this thesis. Other continuous optimization capabilities require first or second derivative information from the user, which not only increases the time required for a cost function evaluation, but also puts a tremendous burden on the user to formulate those extra portions.

A simplex is a geometric figure made up of N+1 points in N dimensions. For example, in two dimensions, a simplex is a triangle. Although the simplex method of linear programming is based on the same geometric concept, it has no relation to the method used here. In general, we are concerned only with simplexes that are nondegenerate, i.e. those that enclose a finite inner N-dimensional volume.

Once the starting simplex is established, the method takes a series of steps attempting to move the simplex to areas of lower function values. At each iteration, a new simplex is constructed by replacing the worst point, i.e., the point at which the function has the highest value. This replacement is accomplished through various geometric manipulations:

1) Reflection of the worst point of the simplex through the face of the other vertices in the simplex.
2) Expansion in the direction along which further decreasing of the cost function is expected.
3) Contraction along one or all dimensions if the reflection increases the cost function.

These basic manipulations are shown for two dimensions in Figure C-1.

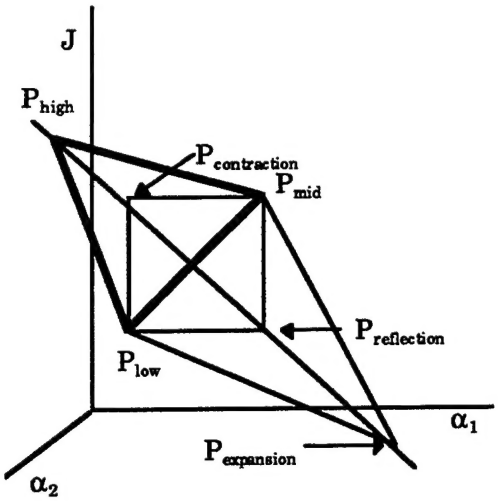


Figure C-1: Simplex modification of the Down-hill Simplex method

Parameter constraints are directly incorporated and non-smooth cost functions can also be used. In some instances the convergence may be slow, typical of techniques not relying on derivative information.

References

[1] Babcock Philip S. An Introduction to Reliability Modeling of Fault Tolerant Systems. Charles Stark Draper Laboratory Report R-1899, September 1986.

[2] Cohon, Jared L. Multiobjective Programming and Planning. San Diego: Academic Press, 1978.

[3] Davis, Lawrence. Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold, 1991.

[4] de Neufville, Richard. Applied Systems Analysis: Engineering Planning and Technology Management. New York: McGraw-Hill, 1990.

[5] Fletcher, R. Practical Methods of Optimization. New York: John Wiley and Sons, 1987.

[6] Genetic Algorithms and Simulated Annealing. Ed. Lawrence Davis. Los Altos: Morgan Kaufmann Publishers, Inc., 1987.

[7] Goldberg, David E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading: Addison-Wesley Publishing Company, Inc., 1989.

[8] Hammett, Robert, Brenan McCarragher, and Andrei Schor. Design Optimizer/Markov Evaluator (DOME) Version 1.0. Charles Stark Draper Laboratory Report R-2409, May 1992.

[9] Howard, Ronald A. Dynamic Programming and Markov Processes. MIT Press, 1960.

[10] Horn, Jeffrey, and Nicholas Nafpliotis. Multiobjective Optimization Using The Niched Pareto Genetic Algorithm. IlliGAL Report No. 93005. University of Illinois at Urbana-Champaign, July 1993.

[11] Michalewicz, Zbigniew and Cezary Z. Janikow. "Handling Constraints in Genetic Algorithms." Proceedings of the Fourth International Conference on Genetic Algorithms. Eds. Richard K. Belew and Lashon B. Booker, San Mateo: Morgan Kaufmann Publishers, Inc., 1991, pp. 151-157.

[12] Multicriteria Design Optimization. Eds. Hans Eschenauer, Juhani Koski, and Andrzej Osyczka, New York: Springer-Verlag, 1990.

[13] Osyczka, Andrzej. Multicriterion Optimization in Engineering. New York: John Wiley and Sons, 1984

[14] Press, Flannery, Teukolsky, and Vetterling. Numerical Recipes: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1986.

[15] Richardson, Jon T., et. al. "Some Guidelines for Genetic Algorithms with Penalty Functions." Proceedings of the Third International Conference on Genetic Algorithms. Ed. J. David Schaffer, Los Altos: Morgan Kaufmann Publishers, Inc., 1989, pp. 191-195.

[16] Rosch, Gene and Andrei L. Schor. TISS Reliability Analysis Presentation. Charles Stark Draper Laboratory, Cambridge, MA. 1 May 1992.

[17] Schaffer, Caruana, Eshelman, and Das. "A study of control parameters affecting on-line performance of genetic algorithms for function optimization." Proceedings of the Third International Conference on Genetic Algorithms. Ed. J. David Schaffer, Los Altos: Morgan Kaufmann Publishers, Inc., 1989, pp. 51-60.

[18] Tillman, Frank, Ching-Lai Hwang, and Way Kuo. Optimization of Systems Reliability. Marcel Dekker, Inc., 1980, pp. 231-238.

[19] Vander Velde, Wallace E. Unpublished notes on a Population Size Rule-of-Thumb. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics. Sept. 1993.

[20] Wright, Stephen and Jorge J Moré. Numerical Optimization Algorithms and Software. Siam Tutorial, 10 May 1992.

[21] Nelder, J.A. and R. Mead. A Simplex Method for Function Minimization. Computer Journal, Vol. 7, 1965.